Near Optimal Coded Data Shuffling for Distributed Learning

Mohamed Adel Attia, *Student Member, IEEE*, and Ravi Tandon, *Senior Member, IEEE*

***Abstract*— Data shuffling between distributed cluster of nodes is one of the critical steps in implementing large-scale learning algorithms. Randomly shuffling the data-set among a cluster of workers allows different nodes to obtain fresh data assignments at each learning epoch. This process has been shown to provide improvements in the learning process (via testing and training error). However, the statistical benefits of distributed data shuffling come at the cost of extra communication overhead from the master node to worker nodes, and can act as one of the major bottlenecks in the overall time for computation. There has been significant recent interest in devising approaches to minimize this communication overhead. One approach is to provision for extra storage at the computing nodes. The other emerging approach is to leverage coded communication to minimize the overall communication overhead. The focus of this work is to understand the fundamental tradeoff between the amount of storage and the communication overhead for distributed data shuffling. In this paper, we first present an information theoretic formulation for the data shuffling problem, accounting for the underlying problem parameters (number of workers, K, number of data points, N, and available storage, and S per node). We then present an information theoretic lower bound on the communication overhead for data shuffling as a function of these parameters. We next present a novel coded communication scheme and show that the resulting communication overhead of the proposed scheme is within a multiplicative factor of at most $\frac{K}{K-1}$ from the lower bound (which is upper bounded by 2 for $K \geq 2$). Furthermore, we present new results towards closing this gap through a novel coded communication scheme, which we call the aligned coded shuffling. This scheme is inspired by the ideas of coded shuffling and interference alignment. In particular, we show that the aligned scheme achieves the optimal storage vs communication trade-off for $K < 5$, and further reduces the maximum multiplicative gap down to $\frac{K-\frac{1}{3}}{K-1}$, for $K \geq 5$.**

***Index Terms*— Coded data shuffling, distributed computing, coded multi-casting, distributed learning.**

I. Introduction

OWING to the parallelized nature of the distributed computing, and the abundance of computational resources over a large cluster of workers, distributed computational frameworks can enable data-intensive learning tasks and big data applications in a timely manner. Distributed computing comes at the unavoidable communication cost due to data transfer to the distributed machines, and the data shuffling process among the distributed workers, which is a basic building block in machine learning paradigms. The data shuffling block can arise in many applications such as: a) random shuffling of the data-set across different points before each learning epoch so that each worker is assigned new training data, which is a common practice that provides statistical benefits, e.g., distributed gradient descent algorithm and its stochastic variations [1]–[4]; b) shuffling the data-set across attributes to assign different features (or attributes) to each worker, e.g., in mobile cloud gaming systems [5]; and c) shuffling the data between the mappers and the reducers in the MapReduce framework [6], where the reducers are interested in collecting the data with the assigned "key(s)" from the mappers.

Another limiting byproduct of distributing the learning process over a large number of machines is the latency caused by the stragglers, i.e., the workers slower than the average due to several factors such as resource contention, disk failure, power limits, and heterogeneous processing capabilities [7], [8]. The straggler problem usually limits the completion time by the slowest worker. Several approaches to mitigate the stragglers effect include a) scheduling redundant computations in [9]–[12], such that any unexpected tardiness or failure of a worker can be compensated by another worker doing the same computations; b) work stealing where the faster workers once they finish their tasks take over the remaining computations from the slower workers [13]; and recently c) work exchange based on the work conservation principle, where coarse heterogenity knowledge/estimation can be used to reassign the work load according to the speed of the workers [14].

A promising research has recently emerged in large scale distributed computing addressing both wired networks, where the computations are done over the cloud [15], [16], and wireless networks, where the computations are done over small mobile machines removing the burden from the cloud [17]–[19]. Distributed computing platforms can also be classified according to the underlying network topology. In the master-worker setting, a centralized master node posses the whole data set and assigns different parts of the data to a set of distributed workers, which collaboratively learn a shared prediction model to be averaged out at the master node later; while in the worker-to-worker setting (also referred to as

Manuscript received January 7, 2018; revised October 26, 2018 and February 22, 2019; accepted June 8, 2019. Date of publication July 3, 2019; date of current version October 18, 2019. This work was supported by the NSF under Grant CAREER-1651492. This paper was presented in part at the 2018 IEEE International Symposium on Information Theory (ISIT), and in part at the 2016 IEEE Global Communications Conference (GLOBECOM).

The authors are with the Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85719 USA (e-mail: madel@email.arizona.edu; tandonr@email.arizona.edu).

Communicated by E. Abbe, Associate Editor for Machine Learning.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2019.2926704

the MapReduce framework [6]), the distributed workers are mapped to train different parts of the data to calculate some functions, then the reducers collect the data with the same "key" to compute each function separately.

The application of coding theory to overcome the communication and latency bottlenecks in order to speed-up the learning process was first considered in [20]. In particular, the idea of using coded data shuffling was first proposed in [20], where excess storage at the workers was utilized to create coded broadcasting opportunities in order to reduce the communication overhead. In the same work, (n, k) Maximum Distance Separable (MDS) codes were proposed for distributed matrix multiplication to mitigate the impact of stragglers. Coded computation using MDS codes in presence of stragglers was proposed in [21] for synchronous gradient descent, and [22]–[24] for linear computation tasks, e.g., matrix multiplication. The use of Polynomial codes for high dimensional coded matrix multiplication was proposed in [25]. Coded computation over wireless networks was proposed in [26], where only one worker can transmit at a time. The use of codes to reduce the communication overhead due to data shuffling was considered in [27]–[36]. In [27]–[30], the authors considered the MapReduce setting, where in order to reduce the communication between the mappers and the reducers, coding opportunities are created with more redundant computations at the mappers, leading to a trade-off between communication and computation. Reference [31], [32] provided a unified coding framework for distributed computing, where the communication load due to shuffling can be alleviated by trading the computational complexity in the presence of straggling servers. The information theoretic limits for data shuffling in the wired master-worker setting was considered in [19], [35], and [36]. Coded data shuffling in wireless setting was recently considered in [19], [35], [36] for both centralized and decentralized approaches.

A. Related Works and Connections to Index Coding

Using codes for random data shuffling over wired master-worker based distributed computational systems was first considered in [20]. A probabilistic coding scheme was introduced showing how using excess storage can reduce the average communication overhead. In our initial preliminary work [33], the optimal worst-case communication overhead was characterized as a function of the available storage for $K = 2, 3$ workers using a systematic storage placement, and data delivery schemes. In another work [34], the no-excess storage case was considered, where it was shown that even for minimum storage value coding opportunities still exist. A systematic coding scheme was developed for any number of workers, which was proven to be information theoretically optimal in the worst-case scenario.

The data shuffling problem can also be viewed as an index coding problem [38], where the amount of data stored at the workers form the side information, and the new data assignments are the messages needed by each worker. The side information in the data shuffling problem is generally not static, where the storage of the workers can be adapted to reduce the communication overhead in the next shuffle. We propose in this work a structural invariant placement (SIP) mechanism, where the storage of the workers is updated according to the latest shuffle to maintain the structure. Furthermore, it was shown in [38] that the index coding is a NP-hard problem, and may require in the worst-case a rate of order $O(K)$, where K is the number of workers. A pliable index coding approach for data shuffling was assumed in [39], where a semi-random shuffles were considered and was shown to achieve a rate of order $O(\log^2(K))$. In this work however, we consider the worst-case rate over all possible shuffles and show that even for the minimum storage (side information at the workers), a rate of order $O(\frac{K-1}{K})$ can be achieved, which does not scale with the number of workers for large values of K.

A recent work in [40] proposed a different coded shuffling scheme based on interference alignment and index coding approaches. The scheme in [40] uses our novel structural invariant placement/update (SIP/SIU) and utilize more coding opportunities. Interestingly, the new coded shuffling scheme in [40] matches our information theoretically lower bound for the worst-case shuffle under the constraint of uncoded cache placement. Moreover, the authors of [40] showed that their coded data shuffling scheme is optimal for any shuffle when the number of files is equal to the number of workers. These results shows that SIP is a key factor in building the coded shuffling scheme.

B. Main Contributions of This Paper

In this paper, we focus on the coded data shuffling for the wired master-worker setting, where coding opportunities are created by exploiting the excess storage at the workers. Before each learning epoch, the data is shuffled at the master node for different training data assignments at each worker, which causes the communication overhead. On one extreme, when all the workers have enough storage to store the whole data set, then no communication is needed for any random shuffle. On the other hand, when the storage is just enough to store the assigned data, which we also refer to as the *no excess storage case*, then the communication is expected to be maximal. Thus, we aim to characterize the fundamental information-theoretic trade-off between the communication overhead due to random shuffling and the available storage at the distributed workers. As a first step towards understanding the fundamental limits of this trade-off for any random shuffle, we focus our attention in this paper on characterizing the optimal trade-off for the worst-case among all shuffles. While it is clear that the average-case rate trade-off will be superior to the worst-case scenario, it is noteworthy that even in the worst-case scenario, there are significant gains from coded data shuffling. Moreover, we show simulations of the performance of our proposed shuffling scheme *averaged over random shuffles* compared to the state-of-the-art schemes. The contributions of this paper are summarized next:

- We first derive an information theoretic lower bound on the worst-case communication overhead for the data shuffling problem. We start by obtaining a family of lower bounds on the rate of some chosen shuffles. Since the rate of any shuffle is at most as large as the worst-case shuffle, the obtained lower bounds serve as valid lower bounds for the worst-case rate as well. We then average out all the lower bounds we get using the chosen shuffles. The key step here is choosing the shuffles which lead to the best lower bound on the communication overhead as a function of the storage. In particular, we consider a set of cyclic shuffles with no overlap between the assigned data batch to any worker in the two subsequent shuffles. Based on a novel bounding methodology similar to the recent results in the caching literature [41] and [42], we are able to express the lower bound as a linear program (LP). We then solve the LP to obtain the best lower bounds on the communication overhead for different regimes of storage.

- Next, we introduce our achievable scheme based on a placement/update procedure that maintains the structure of the storage, which we refer to as *"the structural invariant placement and update (SIP/SIU)"*. The storage placement is inspired by the latest development in the caching literature [43], [44], where we partition the data points across dimensions, which allows each worker to store at least some parts of each data point. Through a careful novel storage update, the structure of the storage can be maintained over time. This allows for applying a data delivery mechanism for any arbitrary shuffle similar to [43], which approaches the optimal communication-storage trade-off in the worst-case (based on the obtained lower bound) within a vanishing gap ratio of $\frac{K}{K-1}$ as the number of distributed workers K increases. It is noteworthy to mention that the storage update mechanism for coded shuffling bears similarities to the process of cache update in online coded caching [44], where the users update their storage according to the change in the popularity pattern. These two cache/storage updates, however, serve different purposes, although the ultimate goal of both update mechanisms is to provide more coding opportunities. In online coded caching, the users update their storage to satisfy the change in the popularity pattern, while our proposed structural invariant update is done to maintain the global structure of the storage across all workers in order to satisfy the data delivery for any subsequent data shuffles.
- Furthermore, we introduce new ideas on how to fully characterize the optimal worst-case communication overhead. We show that by considering more sophisticated interference alignment mechanisms, we can force the interference seen by each worker to occupy the minimum possible dimensions. We refer to this procedure as the *"Aligned Coded Shuffling"* scheme. This scheme also involves a different SIP mechanism of the storage, which we refer to in this paper as the modified structural invariant placement or (modified SIP). The modified SIP mechanism is based on data partitioning and relabeling of the data parts over time. Following these ideas, we can close the gap between the obtained bounds for some storage values, which closes the gap for $K < 5$, and brings the maximum gap ratio down to $\frac{K-\frac{1}{3}}{K-1}$, for $K \geq 5$.
- Finally, we conduct some numerical simulations to compare the *average performance* of our proposed scheme with the random placement scheme in [20]. In our simulations, we do not use the Aligned Coded Shuffling scheme since it is not generalized for any number of workers K. The experimental results show that our scheme outperform the probabilistic scheme in [20] for large storage values with much lower computational complexity. Furthermore, we show the power of our novel modifed SIP mechanism proposed in this paper compared to the random placement in [20]. In our simulations, we use the recent coded shuffling scheme proposed in [40] based on modified SIP for the special case when the number of workers equals to the number of data-points, since the scheme is not generalized yet. The experimental results show that the new coded shuffling scheme in [40] based on modified SIP outperforms the probabilistic scheme in [20] for any value of storage.

C. Notation

The notation $[n_1 : n_2]$ for $n_1 < n_2$, and $n_1, n_2 \in \mathbb{N}$ represents the set of all integers between n_1, and n_2, i.e., $[n_1 : n_2] = \{n_1, n_1 + 1, \ldots, n_2\}$. The combination coefficient $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ equals zero for $k > n$, or $k < 0$. In order to describe subsets of ordered sets, we use the subscript to give the indexes of the elements being chosen from the set, e.g., for the ordered set $\pi = (\pi_1 \ldots, \pi_n)$, $\pi_{[1:4]} = (\pi_1, \pi_2, \pi_3, \pi_4)$. We denote Random Variables (RVs) by capital letters, ordered sets of RVs by capital bold letters, and sets of data points/subpoints by calligraphy letters. The set in the subscript of a set of ordered RVs is used for short notation of a subset of the set of RVs, e.g., for a set of RVs $\mathbf{Z} = \{Z_1, \ldots, Z_n\}$, we use $\mathbf{Z}_{\mathcal{W}}$ to denote the set $\{Z_i\}_{i\in\mathcal{W}}$.

II. System Model

We assume a master node which has access to the entire data-set $\mathcal{A} = \{D_1, D_2, \ldots, D_N\}$ of size Nd bits, i.e., $\mathcal{A}$ is a set containing N data points, denoted by $D_1, D_2, \ldots, D_N$, where d is the dimensionality of each data point. Treating the data points D_n as i.i.d. random variables, we therefore have the entropies of these random variables as

$$H(\mathcal{A}) = N \times H(D_n) = Nd, \quad \forall n \in [1 : N], \tag{1}$$

where $H(.)$ is the entropy function. Table I summarizes the notation used in this paper to denote subsets of the data-set $\mathcal{A}$:

At each iteration, indexed by t, the master node divides the data-set $\mathcal{A}$ into K data batches given as $\mathcal{A}^t(1), \mathcal{A}^t(2), \ldots, \mathcal{A}^t(K)$, where $\mathcal{A}^t(k)$ denotes the data partition designated to be processed by worker w_k at time t, and these batches correspond to the random permutation of the data-set, $\pi^t : \mathcal{A} \to (\mathcal{A}^t(1), \ldots, \mathcal{A}^t(K))$. Note that these data batches are disjoint, and span the whole data-set, i.e.,

$$\mathcal{A}^t(i) \cap \mathcal{A}^t(j) = \phi, \quad \forall i \neq j, \tag{2a}$$

$$\mathcal{A}^t(1) \cup \mathcal{A}^t(2) \cup \ldots \cup \mathcal{A}^t(K) = \mathcal{A}, \quad \forall t. \tag{2b}$$

Hence, the entropy of any batch $\mathcal{A}^t(k)$ is given as

$$H(\mathcal{A}^t(k)) = \frac{1}{K} H(\mathcal{A}) = \frac{N}{K} d, \quad \forall k \in [1 : K]. \tag{3}$$

After getting the data batch, each worker locally computes a function (as an example, this function could correspond to

TABLE I
SUMMARY OF DATA SUBSET NOTATION

Notation	Description	Representation
$\mathcal{A}^t(i)$	The data partition assigned to w_i at iteration t.	-
$\mathcal{A}^t(i,j)$, $i \neq j$	A subset of $\mathcal{A}^t(i)$ stored in the excess storage of w_j at iteration t.	-
$\mathcal{A}^t_j(i)$, $i \neq j$	A subset of $\mathcal{A}^t(i)$ stored *only* in the excess storage of w_j at iteration t.	$\mathcal{A}^t(i,j) \setminus \cup_{k \notin \{i,j\}} \mathcal{A}^t(i,k)$
$\mathcal{A}^t(\mathcal{W})$	The union of the data partitions assigned to every w_i at iteration t, where $i \in \mathcal{W}$.	$\cup_{i \in \mathcal{W}} \mathcal{A}^t(i)$
$\mathcal{A}^t(\mathcal{W},j)$,	The union of the sets $\mathcal{A}^t(i,j)$ for $i \in \mathcal{W}$. $j \notin \mathcal{W}$	$\cup_{i \in \mathcal{W}} \mathcal{A}^t(i,j)$
$\mathcal{A}^t(i,\mathcal{W})$, $i \notin \mathcal{W}$	The union of the sets $\mathcal{A}^t(i,j)$ for $j \in \mathcal{W}$.	$\cup_{j \in \mathcal{W}} \mathcal{A}^t(i,j)$
$\mathcal{A}^t_{\mathcal{W}}(i)$, $i \notin \mathcal{W}$	A subset of $\mathcal{A}^t(i)$ which is *exclusively and jointly* stored in the excess storage at iteration t of all the workers whose indexes are in the set $\mathcal{W}$.	$\cap_{j \in \mathcal{W}} \mathcal{A}^t(i,j) \setminus \cup_{j \notin (\mathcal{W} \cup i)} \mathcal{A}^t(i,j)$

the gradient or sub-gradients of the data points assigned to the worker). The local functions from the K workers are processed subsequently at the master node. We assume that each worker w_k has a storage Z_k^t of size Sd bits, for a real number S, which is used to store some function of the data-set. Therefore, if we consider Z_k^t as a random variable then,

$$H(Z_k^t|\mathcal{A}) = 0, \quad \forall k \in [1:K]. \tag{4}$$

For processing purposes, the assigned data blocks are needed to be stored by the workers, therefore, each worker w_k must at least store the data block $\mathcal{A}^t(k)$ at time t, which gives the storage constraint as

$$H(Z_k^t) = Sd \geq H(\mathcal{A}^t(k)), \qquad \forall k \in [1:K]. \tag{5}$$

According to (3) and (5), we get the *minimum storage per worker* $S \geq \frac{N}{K}$. We also have the *processing constraint* as

$$H(\mathcal{A}^t(k)|Z_k^t) = 0, \quad \forall k \in [1:K], \tag{6}$$

which means $\mathcal{A}^t(k)$ is a deterministic function of the storage Z_k^t.

In the next epoch $t+1$, the data-set is randomly reshuffled at the master node according to a random permutation $\pi^{t+1}: \mathcal{A} \to (\mathcal{A}^{t+1}(1), \mathcal{A}^{t+1}(2), \ldots, \mathcal{A}^{t+1}(K))$, which also satisfies the properties in (2). The main communication bottleneck occurs during *Data Delivery* since the master node needs to communicate the new data batches to the workers. Trivially, if the storage (per worker) exceeds Nd bits, i.e., $S \geq N$, then each worker can store the whole data-set, and no communication has to be done between the master node and the workers for any shuffle. Therefore from the constraint on minimum storage per worker, we can write the possible range for storage as $\frac{N}{K} \leq S \leq N$.

We next proceed to describe the data delivery mechanism, and the associated encoding and decoding functions. The main process can be divided into two phases, namely the data delivery phase and the storage update phase as described next:

A. Data Delivery Phase

At time $t+1$, the master node sends a function of the data batches for the subsequent shuffles (π_t, π_{t+1}), $X_{\pi_t,\pi_{t+1}} = \phi(\mathcal{A}^t(1), \ldots, \mathcal{A}^t(K), \mathcal{A}^{t+1}(1), \ldots, \mathcal{A}^{t+1}(K)) = \phi_{\pi_t,\pi_{t+1}}(\mathcal{A})$ over the shared link, where ϕ is the data delivery encoding function,

$$\phi_{\pi_t,\pi_{t+1}} : \left[2^{\frac{N}{K}d}\right]^{2K} \to [2^{R_{\pi_t,\pi_{t+1}}d}], \tag{7}$$

where $R_{\pi_t,\pi_{t+1}}$ is the rate of the shared link based on the shuffles (π_t, π_{t+1}). Therefore, we have

$$H\left(X_{\pi_t,\pi_{t+1}}|\mathcal{A}\right) = 0, \quad H\left(X_{\pi_t,\pi_{t+1}}\right) = R_{\pi_t,\pi_{t+1}}d, \tag{8}$$

which means that $X_{\pi_t,\pi_{t+1}}$ is a deterministic function of the whole data-set $\mathcal{A}$.

Each worker w_k should reliably decode the desired batch $\mathcal{A}^{t+1}(k)$ out of the transmitted function $X_{\pi_t,\pi_{t+1}}$, as well as the data stored in the previous time slot Z_k^t, i.e., $\mathcal{A}^{t+1}(k) = \psi(X_{\pi_t,\pi_{t+1}}, Z_k^t)$, where ψ is the decoding function at the workers,

$$\psi_{\pi_t,\pi_{t+1}} : [2^{R_{\pi_t,\pi_{t+1}}d}] \times [2^{Sd}] \to [2^{\frac{N}{K}d}]. \tag{9}$$

Therefore, for reliable decoding, we have the following *decodability constraint* at each worker:

$$H\left(\mathcal{A}^{t+1}(k)|Z_k^t, X_{\pi_t,\pi_{t+1}}\right) = 0, \quad \forall k \in \{1, \ldots, K\}. \tag{10}$$

B. Storage Update Phase

At the next iteration $t+1$, every worker updates its stored content according to the placement strategy, where the new storage content for worker w_k is given by Z_k^{t+1}, which is a function of the old storage content Z_k^t as well as transmitted function $X_{\pi_t,\pi_{t+1}}$, i.e., $Z_k^{t+1} = \mu(X_{\pi_t,\pi_{t+1}}, Z_k^t)$, where μ is the update function

$$\mu_{\pi_t,\pi_{t+1}} : [2^{R_{\pi_t,\pi_{t+1}}d}] \times [2^{Sd}] \to [2^{Sd}], \tag{11}$$

Therefore, we have the following *storage-update constraint*:

$$H(Z_k^{t+1}|Z_k^t, X_{\pi_t,\pi_{t+1}}) = 0, \quad \forall k \in \{1, \ldots, K\}. \tag{12}$$

The excess storage after storing $\mathcal{A}^{t+1}(k)$ in Z_k^{t+1}, given by $\left(S - \frac{N}{K}\right) d$ bits, can be used to store opportunistically a function of the remaining $K-1$ data batches. For the scope of this work, we assume that the placement of the excess storage is uncoded, which means that the excess storage is dedicated to store uncoded functions of the remaining $K-1$ batches. We give the notation $\mathcal{A}^{t+1}(i,k)$, where $i \neq k$, as the part of data that worker w_k stores about $\mathcal{A}^{t+1}(i)$ in the excess storage at time $t+1$. As a result, we can write the content of Z_k^{t+1} for uncoded storage placement as

$$Z_k^{t+1} = \left\{ \mathcal{A}^{t+1}(k), \bigcup_{j \in [1:K] \setminus k} \mathcal{A}^{t+1}(j,k) \right\}. \quad (13)$$

Furthermore, we assume a generic placement strategy for the excess storage as follows: the batch $\mathcal{A}^{t+1}(i)$ consists of 2^{K-1} partitions, denoted as $\mathcal{A}_{\mathcal{W}}^{t+1}(i)$, $\mathcal{W} \in 2^{[1:K]\setminus i}$, where $2^{[1:K]\setminus i}$ is the power set of all possible subsets of the set $[1:K] \setminus i$ including the empty set. The worker w_k, for $k \neq i$, stores the partition $\mathcal{A}_{\mathcal{W}}^{t+1}(i)$ in the excess storage, only if $k \in \mathcal{W}$. Therefore, the sub-batches $\mathcal{A}^{t+1}(i)$, and $\mathcal{A}^{t+1}(i,k)$ can be expressed as

$$\begin{aligned} \mathcal{A}^{t+1}(i) &= \bigcup_{\mathcal{W} \subseteq [1:K]\setminus i} \mathcal{A}_{\mathcal{W}}^{t+1}(i), \\ \mathcal{A}^{t+1}(i,k) &= \bigcup_{\mathcal{W} \subseteq [1:K]\setminus i:\, k \in \mathcal{W}} \mathcal{A}_{\mathcal{W}}^{t+1}(i). \end{aligned} \quad (14)$$

Let us consider $\mathcal{A}_{\mathcal{W}}^{t+1}(i)$ as a random variable with entropy $H(\mathcal{A}_{\mathcal{W}}^{t+1}(i)) = |\mathcal{A}_{\mathcal{W}}^{t+1}(i)|d$, and size $|\mathcal{A}_{\mathcal{W}}^{t+1}(i)|$ normalized by the data point size d. Therefore, the following two constraints are obtained:

- **Data size constraint:** The first constraint is related to the total size of the data given by Nd bits,

$$\begin{aligned} N &= \frac{1}{d} H(\mathcal{A}) = \frac{1}{d} \sum_{i=1}^{K} H(\mathcal{A}^{t+1}(i)) \\ &\overset{(a)}{=} \frac{1}{d} \sum_{i=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]\setminus i} H(\mathcal{A}_{\mathcal{W}}^{t+1}(i)) \\ &= \sum_{\ell=1}^{K} \sum_{i=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]\setminus i:\, |\mathcal{W}|=\ell} |\mathcal{A}_{\mathcal{W}}^{t+1}(i)| = \sum_{\ell=1}^{K} x_\ell, \end{aligned} \quad (15)$$

where (a) follows from (14), and $x_\ell \geq 0$ is defined as

$$x_\ell \overset{\Delta}{=} \sum_{i=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]\setminus i:\, |\mathcal{W}|=\ell} |\mathcal{A}_{\mathcal{W}}^{t}(i)|, \quad \ell \in [0:K-1]. \quad (16)$$

- **Excess storage size constraint:** The second constraint is related to the total excess storage of all the workers, which cannot exceed $K\left(S - \frac{N}{K}\right) d$ bits,

$$\begin{aligned} K\left(S - \frac{N}{K}\right) &\geq \frac{1}{d} \sum_{i=1}^{K} \sum_{k \in [1:K]\setminus i} H\left(\mathcal{A}^{t+1}(i,k)\right) \\ &\overset{(a)}{=} \sum_{i=1}^{K} \sum_{k \in [1:K]\setminus i} \sum_{\mathcal{W} \subseteq [1:K]\setminus i:\, k \in \mathcal{W}} |\mathcal{A}_{\mathcal{W}}^{t+1}(i)| \\ &\overset{(b)}{=} \sum_{i=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]\setminus i} |\mathcal{W}|\, |\mathcal{A}_{\mathcal{W}}^{t+1}(i)| \\ &= \sum_{\ell=1}^{K} \ell \sum_{i=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]\setminus i:\, |\mathcal{W}|=t} |\mathcal{A}_{\mathcal{W}}^{t+1}(i)| \overset{(c)}{=} \sum_{\ell=1}^{K} \ell x_\ell, \end{aligned} \quad (17)$$

where (a) follows from (14), (b) is true because when we sum up the contents of the excess storage at all the workers, the chunk $\mathcal{A}_{\mathcal{W}}^{t+1}(i)$ is counted $|\mathcal{W}|$ number of times, which is the number of workers storing this chunk, and (c) follows from (16).

A data shuffling scheme is characterized by the data delivery encoding function $\phi_{\pi_t,\pi_{t+1}}$, decoding function $\psi_{\pi_t,\pi_{t+1}}$, and storage update function $\mu_{\pi_t,\pi_{t+1}}$ defined in (7), (9), and (11), respectively. Note that the functions $(\phi_{\pi_t,\pi_{t+1}}, \psi_{\pi_t,\pi_{t+1}}, \mu_{\pi_t,\pi_{t+1}})$ depend on the shuffle (π_t, π_{t+1}). For ease of notation, we drop (π_t, π_{t+1}) in the following discussion. We next define the worst-case communication as follows:

*Definition 1 (**Worst-Case Communication**):* For any achievable scheme characterized by the encoding, decoding, and cache update functions (ϕ, ψ, μ), the worst-case communication overhead over all possible consecutive data shuffles (π_t, π_{t+1}) is defined as

$$R_{\text{worst-case}}^{(\phi,\psi,\mu)}(S) \overset{\Delta}{=} \max_{(\pi_t,\pi_{t+1})} R_{(\pi_t,\pi_{t+1})}^{(\phi,\psi,\mu)}(S). \quad (18)$$

Our goal in this work is to characterize the optimal worst-case communication $R_{\text{worst-case}}^{*}(K,N,S)$ defined as

$$R_{\text{worst-case}}^{*}(S) \overset{\Delta}{=} \min_{(\phi,\psi,\mu)} R_{\text{worst-case}}^{(\phi,\psi,\mu)}(S). \quad (19)$$

We next present a claim which shows that the optimal worst-case communication $R_{\text{worst-case}}^{*}(S)$ is a convex function of the storage S:

Claim 1: $R_{\text{worst-case}}^{*}(S)$ is a convex function of S, where S is the available storage at each worker.

Proof: Claim 1 follows from a simple memory sharing argument which shows that for any two available storage values S_1 and S_2, if $(S_1, R_{\text{worst-case}}^{*}(S_1))$, and $(S_2, R_{\text{worst-case}}^{*}(S_2))$ are achievable optimal schemes, then for any storage $\bar{S} = \alpha S_1 + (1-\alpha) S_2$, $0 \leq \alpha \leq 1$, there is a scheme which achieves a communication overhead of $\bar{R}(\bar{S}) = \alpha R_{\text{worst-case}}^{*}(S_1) + (1-\alpha) R_{\text{worst-case}}^{*}(S_2)$.

This is done as follows: first, we divide the data-set $\mathcal{A}$ across d dimensions into 2 batches namely; $\mathcal{A}^{(\alpha)}$, and $\mathcal{A}^{(1-\alpha)}$ of dimensions αd, and $(1-\alpha)d$, for each point respectively. Then, we divide the storage for every worker w_k into 2 parts namely; $Z_k^{(\alpha)}$, and $Z_k^{(1-\alpha)}$ of size $S_1 \alpha d$, and $S_2(1-\alpha)d$, respectively. The former batch $\mathcal{A}^{(\alpha)}$ will be shuffled among the former part of the storage $Z_k^{(\alpha)}$ to achieve the point $(S_1, R_{\text{worst-case}}^{*}(S_1))$, while the latter batch $\mathcal{A}^{(1-\alpha)}$ will be shuffled among the latter part of the storage $Z_k^{(1-\alpha)}$ to achieve the point $(S_2, R_{\text{worst-case}}^{*}(S_2))$. Therefore, the total achievable load is given by

$$\begin{aligned} H(X) &= R_{\text{worst-case}}^{*}(S_1)\alpha d + R_{\text{worst-case}}^{*}(S_2)(1-\alpha)d \\ &= \bar{R}(\bar{S})d. \end{aligned} \quad (20)$$

We next note that the optimal communication rate $R^*_{\text{worst-case}}(\bar{S})$ is upper bounded by $\bar{R}(\bar{S})$, the rate of the memory sharing scheme, i.e.,

$$R^*_{\text{worst-case}}(\alpha S_1 + (1-\alpha)S_2) \leq \alpha R^*_{\text{worst-case}}(S_1) + (1-\alpha)R^*_{\text{worst-case}}(S_2), \quad (21)$$

which shows that $R^*_{\text{worst-case}}(S)$ is a convex function of S. ■

III. Main Results and Discussions

A. Theorem 1: Achievablitiy in the Worst-Case

The first theorem presents an achievable worst-case rate $R_{\text{worst-case}}$, which also yields an upper bound on the optimal storage-rate trade-off $R^*_{\text{worst-case}}$.

Theorem 1: For a data-set containing $N \in \mathbb{N}$ data points, and a set of $K \in \mathbb{N}$ distributed workers, the lower convex envelope of the following $K+1$ storage-rate pairs is achievable for all $i \in [0:K]$:

$$\left(S = \left(1 + i\frac{K-1}{K}\right)\frac{N}{K},\ R^{\text{upper}}_{\text{worst-case}} = \frac{N(K-i)}{K(i+1)}\right). \quad (22)$$

The proof of Theorem 1 is presented in Appendix I. We present an encoding, decoding, and cache update scheme, which achieves the above rate-storage pairs. One of the crucial steps in the proof is the *structural invariant placement and update (SIP/SIU)* of the storage of the workers over time. The storage placement involves partitioning the data points across dimensions, which allows each worker to store at least some parts of each data point, which in turns introduces a local storage gain for any potential data assignment. In order to increase the global gain through increasing the coding opportunities, we minimize the overlap between the parts stored by each worker of each data point. Through a careful novel update of storage across time, the structure can be maintained for any random data assignment, which allows applying a coded data delivery mechanism to reduce the communication overhead. Now, we give the following illustrative example for $K = N = 4$ to introduce the main elements of the achievability proof.

1) Example 1: Achievability for $N=4$ and $K=4$: Consider the case of $K = 4$ workers, and $N = 4$ i.i.d. data points, labeled as $\{D_1, D_2, D_3, D_4\}$. According to Theorem 1, the achievable worst-case storage-rate trade-off is given by the lower convex envelope of the 5 storage-rate pairs ($S = 3i/4 + 1$, $R = (4-i)/(i+1)$) for $i \in [0:4]$, which is also shown by the red dashed curve in Figure 1.

From Claim 1, once we achieve these pairs, the lower convex envelope is also achievable by memory sharing. At time t, we consider the data is assigned according to the shuffle $\pi_t = (1,2,3,4)$, e.g., ω_1 is assigned the data point D_1, i.e., $\mathcal{A}^t(1) = D_1$. At time $t+1$, we consider the cyclic shuffle $\pi_{t+1} = (2,3,4,1)$, e.g., ω_1 is assigned the data point D_2 at time $t+1$, i.e., $\mathcal{A}^{t+1}(1) = D_2$. Once we achieve the rate for the shuffle $\pi_{t+1} = (2,3,4,1)$, a similar data delivery mechanism can be used for any $\pi_{t+1} \in [4!]$, where $[4!]$ is the set containing all the 4! possible permutations of the set $[1:4]$. The achievability, according to (π_t, π_{t+1}), for the storage value $S = 3i/4 + 1$ and $i \in [0:4]$ follows next.

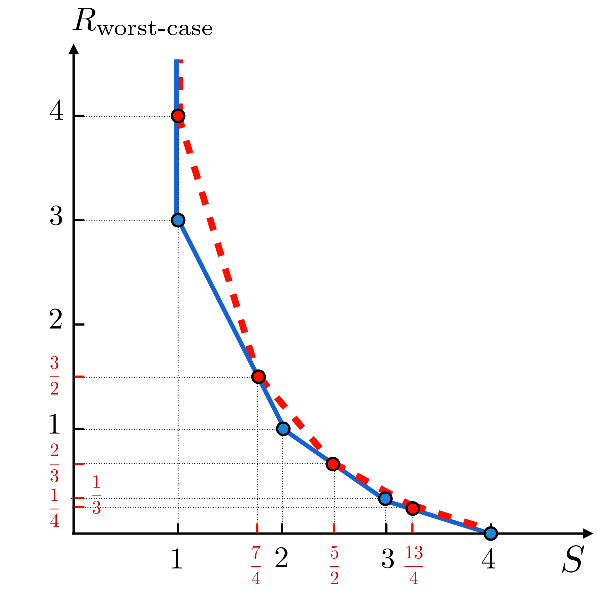

Fig. 1. The lower bound and the upper bound on the worst-case rate $R^*_{\text{worst-case}}$ for $N = 4$, and $K = 4$ versus the amount of storage S. The maximum gap appears to be when $S = 1$, which is given as a ratio 4/3.

- **Case $\mathbf{i = 0}$ ($\mathbf{S = 1}$):**

This storage value represents the no-excess storage case, where every worker only stores the assigned data point under processing. To satisfy the new assignment at time $t+1$, we choose now to send the 4 data points, which satisfies any shuffle at time $t+1$, achieving the pair ($S = 1, R = 4$). Later in Appendix IV, we will show how to improve this rate and prove that in fact ($S = 1, R = 3$) is optimal. The storage update is trivial in this case, where every worker keeps the new assigned data point and discard the remaining three points.

- **Case $\mathbf{i = 1}$ ($\mathbf{S = 7/4}$):**

Storage Placement: The storage placement for $i = 1$ is shown in Figure 2a. First, every data point is partitioned into 4 sub-points of size $d/4$ bits each, where every sub-point is labeled by a unique subset $\mathcal{W} \subseteq [1:4]$ of size $|\mathcal{W}| = 1$. For instance, the data point D_1 is partitioned as follows:

$$D_1 = \{D_{1,\{1\}}, D_{1,\{2\}}, D_{1,\{3\}}, D_{1,\{4\}}\}. \quad (23)$$

Every worker first fully stores the assigned data point. For the excess storage, every worker ω_k stores from the remaining points, not being processed, the sub-points where $k \in \mathcal{W}$. For instance, ω_1 stores 1 sub-point of D_2, labeled as $\mathcal{A}^t(2,1) = \{D_{2,\{1\}}\}$. To summarize, each worker stores the assigned data point of size d, and for each one of the remaining 3 data points, it stores 1 sub-point of size $d/4$. That is, the storage requirement is given by $S = 1 + 3 \times 1/4 = 7/4$, which satisfies the storage constraint for $i = 1$ ($S = 7/4$).

Data Delivery: According to the storage placement at time t in Figure 2a, at time $t+1$ every worker needs 3 sub-points of the assigned data point, and every sub-point is available at least in one of the remaining workers, e.g., ω_1 needs the sub-points $\{D_{2,\{2\}}, D_{2,\{3\}}, D_{2,\{4\}}\}$. Now, if we pick any 2 out of the 4 workers, then each one of the 2 workers needs a sub-point available at the other worker. Therefore, we can send an *"order 2"* symbol, of size $d/4$ bits, useful for these chosen

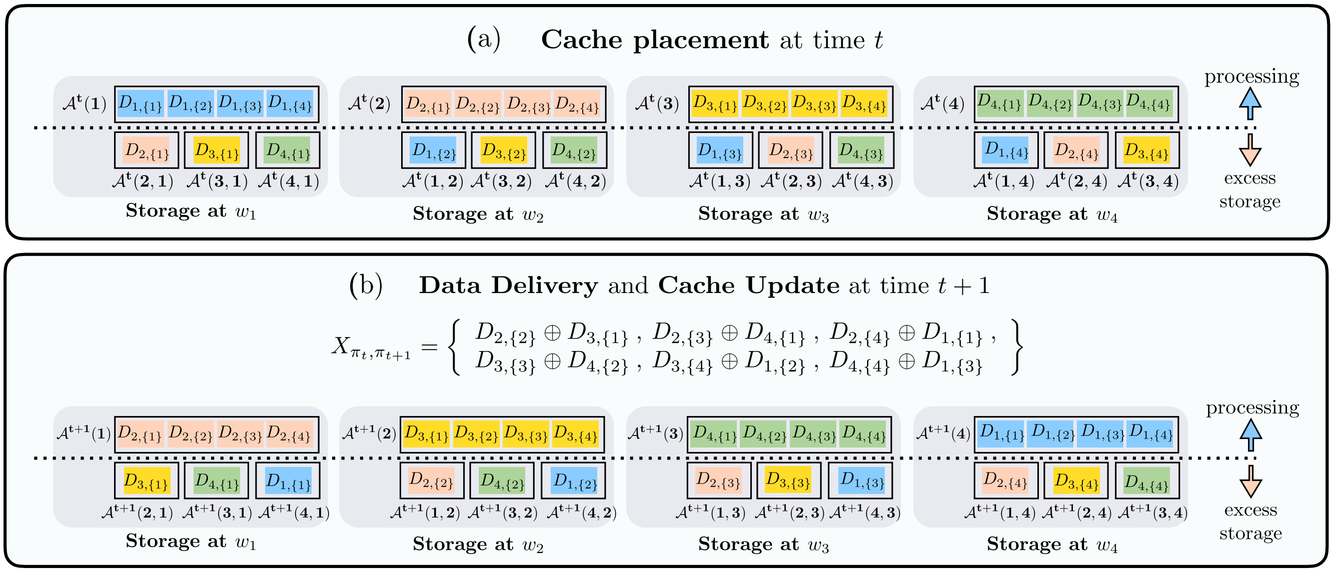

Fig. 2. Structural invariant storage placement (SIP), (a), and update, (b), for $K = 4$ workers, $N = 4$ data points, and $i = 1$ ($S = 7/4$). Every data point is partitioned into 4 sub-points each labeled by a unique subset of the set $[1:4]$ of length 1. Above the dotted line is the data point fully stored for processing, and below the dotted line is the excess storage used to store the sub-points containing the worker's index.

two workers in the same time, and for all possible choices of 2 out of the 4 workers we send the following $\binom{4}{2} = 6$ coded symbols which satisfies the required 3 needed sub-points for the 4 workers:

$$X_{\pi_t,\pi_{t+1}} = \begin{Bmatrix} D_{2,\{2\}} \oplus D_{3,\{1\}}, & \text{useful for } w_1, w_2, \\ D_{2,\{3\}} \oplus D_{4,\{1\}}, & \text{useful for } w_1, w_3, \\ D_{2,\{4\}} \oplus D_{1,\{1\}}, & \text{useful for } w_1, w_4, \\ D_{3,\{3\}} \oplus D_{4,\{2\}}, & \text{useful for } w_2, w_3, \\ D_{3,\{4\}} \oplus D_{1,\{2\}}, & \text{useful for } w_2, w_4, \\ D_{4,\{4\}} \oplus D_{1,\{3\}}, & \text{useful for } w_3, w_4 \end{Bmatrix}. \tag{24}$$

The rate of this transmission is $\binom{4}{2}/4 = 3/2$, and the pair $(S = 7/4, R = 3/2)$ is achieved.

Storage Update: At time $t+1$, the storage update follows from Figure 2b. In order to maintain the structure of the storage, the workers first store the data points newly assigned and acquired from the delivery phase. For the excess storage update, each worker w_k keeps from the data point previously assigned at time t the sub-points which are labeled by a set $\mathcal{W}$ where $k \in \mathcal{W}$. For example, w_1 keeps from $\mathcal{A}^t(1) = \mathcal{A}^{t+1}(4) = D_1$ the sub-point $\mathcal{A}^{t+1}(4,1) = \{D_{1,\{1\}}\}$.

- **Case $\mathbf{i = 2}$ ($\mathbf{S = 5/2}$):**

Storage Placement: The storage placement for $i = 2$ is shown in Figure 3a. First, every data point is partitioned into 6 sub-points of size $d/6$ bits each, where every sub-point is labeled by a unique subset $\mathcal{W} \subseteq [1:4]$ of size $|\mathcal{W}| = 2$. For instance, the data point D_1 is partitioned as follows:

$$D_1 = \{D_{1,\{1,2\}}, D_{1,\{1,3\}}, D_{1,\{1,4\}}, D_{1,\{2,3\}}, D_{1,\{2,4\}}, D_{1,\{3,4\}}\}. \tag{25}$$

Every worker first fully stores the assigned data point. For the excess storage, every worker w_k stores from the remaining points, not being processed, the sub-points where $k \in \mathcal{W}$. For instance, w_1 stores 3 sub-point of D_2, labeled as $\mathcal{A}^t(2,1) = \{D_{2,\{1,2\}}, D_{2,\{1,3\}}, D_{2,\{1,4\}}\}$. To summarize, each worker stores the assigned data point of size d, and for each one of the remaining 3 data points, it stores 3 sub-point of size $d/6$ each. That is, the storage requirement is given by $S = 1 + 3 \times 3 \times 1/6 = 5/2$, which satisfies the storage constraint for $i = 2$ ($S = 5/2$).

Data Delivery: According to the storage placement at time t in Figure 3a, at time $t+1$ every worker needs 3 sub-points of the assigned data point, and every sub-point is available at least in two of the remaining workers, e.g., w_1 needs the sub-points $\{D_{2,\{2,3\}}, D_{2,\{2,4\}}, D_{2,\{3,4\}}\}$. Now, if we pick any 3 out of the 4 workers, then every one of the 3 workers needs a sub-point available at the other 2 workers. Therefore, we can send an order 3 symbol, of size $d/6$ bits, useful for these chosen workers in the same time, and for all possible choices of 3 out of the 4 workers we send the following $\binom{4}{3} = 4$ coded symbols which satisfies the required 3 needed sub-points for the 4 workers:

$$X_{\pi_t,\pi_{t+1}} = \begin{Bmatrix} D_{2,\{2,3\}} \oplus D_{3,\{1,3\}} \oplus D_{4,\{1,2\}}, & \text{useful for } w_1, w_2, w_3, \\ D_{2,\{2,4\}} \oplus D_{3,\{1,4\}} \oplus D_{1,\{1,2\}}, & \text{useful for } w_1, w_2, w_4, \\ D_{2,\{3,4\}} \oplus D_{4,\{1,4\}} \oplus D_{1,\{1,3\}}, & \text{useful for } w_1, w_3, w_4, \\ D_{3,\{3,4\}} \oplus D_{4,\{2,4\}} \oplus D_{1,\{2,3\}}, & \text{useful for } w_2, w_3, w_4 \end{Bmatrix}.$$

The rate of this transmission is $\binom{4}{3}/\binom{4}{2} = 2/3$, and the pair $(S = 5/2, R = 2/3)$ is achieved.

Storage Update: At time $t+1$, the storage update follows from Figure 3b. In order to maintain the structure of the storage, the workers first store the data points newly assigned and acquired from the delivery phase. For the excess storage update, each worker w_k keeps from the data point previously assigned at time t the sub-points which are

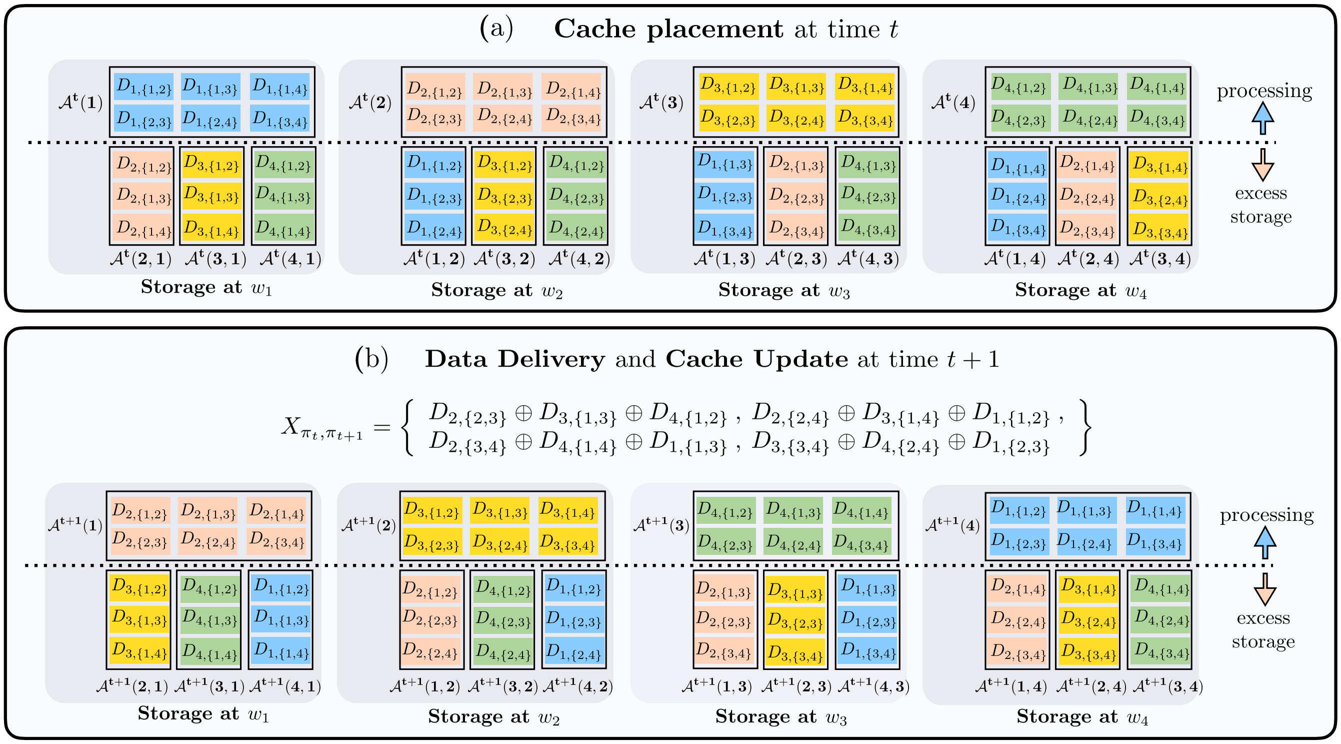

Fig. 3. Structural invariant storage placement (SIP), (a), and update, (b), for $K = 4$ workers, $N = 4$ data points, and $i = 2$ ($S = 5/2$). Every data point is partitioned into 6 sub-points each labeled by a unique subset of the set $[1:4]$ of length 2. Above the dotted line is the data point fully stored for processing, and below the dotted line is the excess storage used to store the sub-points containing the worker's index.

labeled by a set $\mathcal{W}$ where $k \in \mathcal{W}$. For example, w_1 keeps from $\mathcal{A}^t(1) = \mathcal{A}^{t+1}(4) = D_1$ the sub-point $\mathcal{A}^{t+1}(4,1) = \{D_{1,\{1,2\}}, D_{1,\{1,3\}}, D_{1,\{1,4\}}\}$.

- **Case i = 3 (S = 13/4)**:

Storage Placement: The storage placement for $i = 3$ is shown in Figure 4a. First, every data point is partitioned into 4 sub-points of size $d/4$ bits each, where every sub-point is labeled by a unique subset $\mathcal{W} \subseteq [1:4]$ of size $|\mathcal{W}| = 3$. For instance, the data point D_1 is partitioned as follows:

$$D_1 = \{D_{1,\{1,2,3\}}, D_{1,\{1,2,4\}}, D_{1,\{1,3,4\}}, D_{1,\{2,3,4\}}\}. \tag{26}$$

Every worker first fully stores the assigned data point. For the excess storage, every worker w_k stores from the remaining points, not being processed, the sub-points where $k \in \mathcal{W}$. For instance, w_1 stores 3 sub-point of D_2, labeled as $\mathcal{A}^t(2,1) = \{D_{2,\{1,2,3\}}, D_{2,\{2,2,4\}}, D_{2,\{1,3,4\}}\}$. To summarize, each worker stores the assigned data point of size d, and for each one of the remaining 3 data points, it stores 3 sub-point of size $d/4$ each. That is, the storage requirement is given by $S = 1+3 \times 3 \times 1/4 = 13/4$, which satisfies the storage constraint for $i = 3$ ($S = 13/4$).

Data Delivery: According to the storage placement at time t in Figure 4a, at time $t+1$ every worker only needs one sub-point of the assigned data point which is available at the three remaining workers, e.g., w_1 needs $D_{2,\{2,3,4\}}$ which is available at the workers w_2, w_3, and w_4. Therefore, we can send the following order 4 symbol useful for all the 4 workers at the same time:

$$X_{\pi_t,\pi_{t+1}} = \{D_{2,\{2,3,4\}} \oplus D_{3,\{1,3,4\}} \oplus D_{4,\{1,2,4\}} \oplus D_{1,\{1,2,3\}}\}. \tag{27}$$

Since the the size of each sub-point is $d/4$, the rate of the transmission is $\binom{4}{4}/4 = 1/4$ and hence the pair ($S = 13/4, R = 1/4$) is achieved.

Storage Update: At time $t+1$, the storage update follows from Figure 4b. In order to maintain the structure of the storage, the workers first store the data points newly assigned and acquired from the delivery phase. For the excess storage update, each worker w_k keeps from the data point previously assigned at time t the sub-points which are labeled by a set $\mathcal{W}$ where $k \in \mathcal{W}$. For example, w_1 keeps from $\mathcal{A}^t(1) = \mathcal{A}^{t+1}(4) = D_1$ the sub-point $\mathcal{A}^{t+1}(4,1) = \{D_{1,\{1,2,3\}}, D_{1,\{2,2,4\}}, D_{1,\{1,3,4\}}\}$.

- **Case i = 4 (S = 4)**: This case is trivial where every worker can store all the 4 data points and hence no communication is needed for any shuffle. Therefore, the pair ($S = 4, R = 0$) is achieved.

B. Theorem 2: Lower Bounds on the Optimal Worst-Case Rate

We next present our second main result in Theorem 2, which gives an information theoretic lower bound on the optimal worst-case rate.

Theorem 2: For a data-set containing $N \in \mathbb{N}$ data points, and a set of $K \in \mathbb{N}$ distributed workers, a lower bound

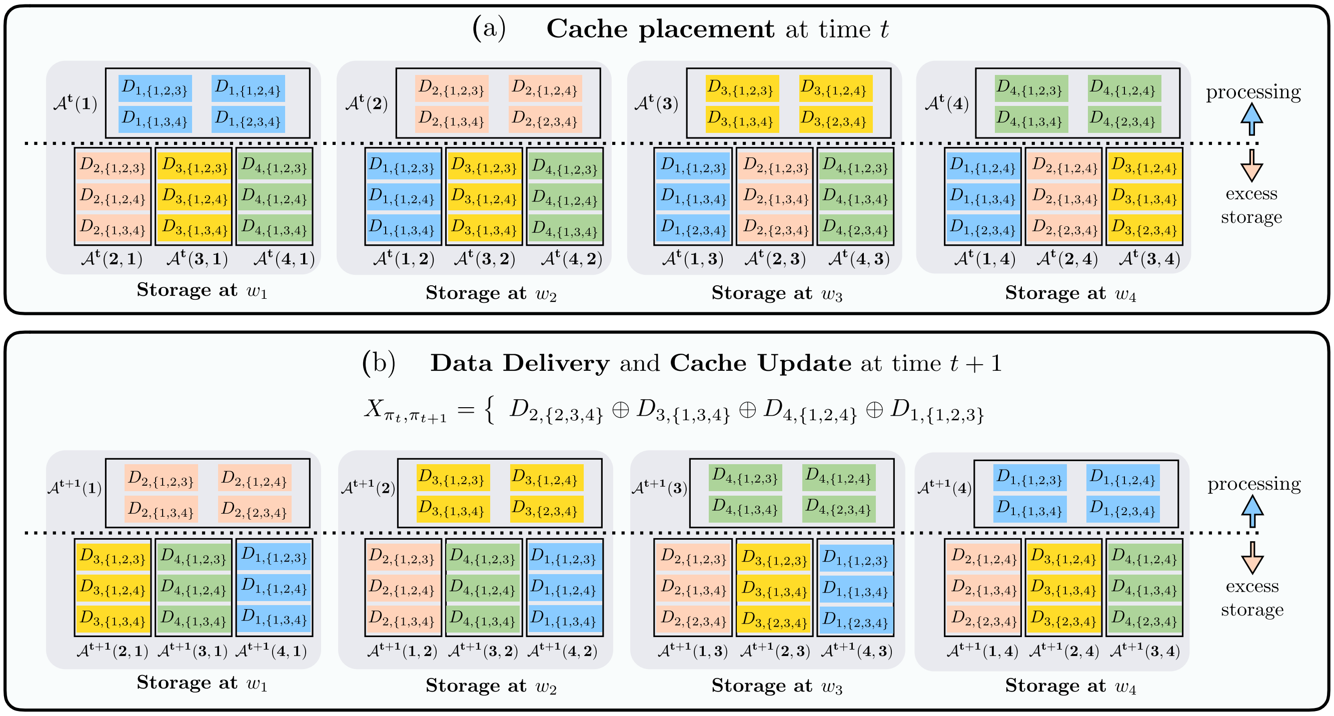

Fig. 4. Structural invariant storage placement (SIP), (a), and update, (b), for $K = 4$ workers, $N = 4$ data points, and $i = 3$ ($S = 13/4$). Every data point is partitioned into 4 sub-points each labeled by a unique subset of the set $[1:4]$ of length 3. Above the dotted line is the data point fully stored for processing, and below the dotted line is the excess storage used to store the sub-points containing the worker's index.

on $R^*_{\text{worst-case}}$ is given by the lower convex envelope of the following K storage-rate pairs:

$$\left(S = m\frac{N}{K},\ R^{\text{lower}}_{\text{worst-case}} = \frac{N(K-m)}{Km}\right),\ \forall m \in [1:K]. \quad (28)$$

The complete proof of Theorem 2 can be found in Appendix II.

*Remark 1 (**Basic idea for the lower bound**):* A lower bound on the optimal rate $R^*_{\pi_t,\pi_{t+1}}$ of a shuffle (π_t, π_{t+1}) serves also as a lower bound on the worst-case since the optimal worst-case rate is larger than the optimal rate for any shuffle, i.e., $R^*_{\text{worst-case}} \geq R^*_{\pi_t,\pi_{t+1}}$. Therefore, we get lower bounds over $R^*_{\text{worst-case}}$ by focusing on a sequence of shuffles, and then average out all the lower bounds. The novel part in our proof is to carefully choose the right shuffles which lead to the best lower bound.

In our converse proof, we apply a novel bounding methodology similar to the recent result in [41], [42], where the optimal uncoded cache placement problem for a file delivery system is considered. In this paper however, we consider the data delivery based on subsequent assignments according to random shuffles of the data. Our problem also requires storing the data under processing, and allows for storage update over time as opposed to [41], [42]. At the end, we arrive at a linear program subject to the problem constraints (data size and storage constraints), which can be solved to obtain the best lower bounds over different regimes of the available storage. In the following example, we show how to obtain the lower bounds on the worst-case rate for the case of $N = K = 4$.

1) Example 2: Lower Bounds for $N = 4$ and $K = 4$: Consider the case of $K = 4$ workers, and $N = 4$ i.i.d. data points, labeled as $\{D_1, D_2, D_3, D_4\}$. Assume the 4 data points are assigned at time t according to $\pi_t = (1, 2, 3, 4)$, i.e., $\mathcal{A}^t(k) = D_k$ for $k \in [1:4]$. Therefore, at time t, the data point D_k is fully stored at the cache of the worker w_k, and partially stored at the remaining workers, which gives the storage content of the worker w_k as follows:

$$Z_k^{t+1} = \left\{D_k,\ \bigcup_{j\in[1:4]\setminus k} D_j(k)\right\}, \quad (29)$$

where $D_j(k)$ is the part of D_j stored in the excess storage of worker w_k at time t.

We start by considering the following shuffle (π_t, π_{t+1}): for a permutation $\sigma : (1, 2, 3, 4) \to (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$, the worker w_{σ_k} is assigned at time $t+1$ the data point that was assigned to the worker $w_{\sigma_{k-1}}$ at time t, i.e., $\mathcal{A}^{t+1}(\sigma_k) = \mathcal{A}^t(\sigma_{k-1}) = D_{\sigma_{k-1}}$. Using the decodability constraint in (10), worker w_{σ_k} must be able to decode $\mathcal{A}^{t+1}(\sigma_k) = D_{\sigma_{k-1}}$ using its own cache $Z^t_{\sigma_k}$ as well as the transmission $X_{(\pi_t,\pi_{t+1})}$ which gives the following condition:

$$\begin{aligned} &H(\mathcal{A}^{t+1}(\sigma_k)|Z^t_{\sigma_k}, X_{(\pi_t,\pi_{t+1})}) \\ &= H(D_{\sigma_{k-1}}|Z^t_{\sigma_k}, X_{(\pi_t,\pi_{t+1})}) = 0,\ \forall k \in [1:4]. \end{aligned} \quad (30)$$

Furthermore, from (6), each worker should store the assigned data point at time t, therefore,

$$H(\mathcal{A}^t(\sigma_k)|Z^t_{\sigma_k}) = H(D_{\sigma_k}|Z^t_{\sigma_k}) = 0,\ \forall k \in [1:4]. \quad (31)$$

Note that the conditions (30) and (31) fully characterize the shuffle (π_t, π_{t+1}). Consequently, the transmission $X_{\pi_t,\pi_{t+1}}$ as well as the cache of any three workers can decode the 4 data points, which can be shown as follows:

$$\begin{aligned} &H(\mathcal{A}|\mathbf{Z}^t_{\sigma[2:4]}, X_{(\pi_t,\pi_{t+1})}) \\ &\quad= H(D_1, D_2, D_3, D_4|Z^t_{\sigma[2:4]}, X_{(\pi_t,\pi_{t+1})}) \\ &\quad= H(D_{\sigma_1}, D_{\sigma_2}, D_{\sigma_3}, D_{\sigma_4}|Z^t_{\sigma[2:4]}, X_{(\pi_t,\pi_{t+1})}) \\ &\quad\overset{(a)}{\le} H(D_{\sigma_1}|Z^t_{\sigma_2}, X_{(\pi_t,\pi_{t+1})}) + H(D_{\sigma_2}|Z^t_{\sigma_2}) \\ &\qquad + H(D_{\sigma_3}|Z^t_{\sigma_3}) + H(D_{\sigma_4}|Z^t_{\sigma_4}) \overset{(b)}{=} 0, \end{aligned} \tag{32}$$

where (a) follows from the fact that $H(A, B) \le H(A) + H(B)$ and that conditioning reduces entropy, and (b) follows directly using (30) and (31). Next, we obtain the following bound using (32):

$$\begin{aligned} 4d &= H(\mathcal{A}) \\ &= I(A; \mathbf{Z}^t_{\sigma[2:4]}, X_{\pi_t,\pi_{t+1}}) + H(\mathcal{A}|\mathbf{Z}^t_{\sigma[2:4]}, X_{\pi_t,\pi_{t+1}}) \\ &\overset{(a)}{=} I(A; \mathbf{Z}^t_{\sigma[2:4]}, X_{\pi_t,\pi_{t+1}}) \overset{(b)}{=} H(\mathbf{Z}^t_{\sigma[2:4]}, X_{\pi_t,\pi_{t+1}}) \\ &\overset{(c)}{=} H(Z^t_{\sigma_4}, X_{\pi_t,\pi_{t+1}}) + H(Z^t_{\sigma_2}, Z^t_{\sigma_3}|Z^t_{\sigma_4}, X_{\pi_t,\pi_{t+1}}) \\ &\le H(X_{\pi_t,\pi_{t+1}}) + H(Z^t_{\sigma_4}) + H(Z^t_{\sigma_3}|Z^t_{\sigma_4}, X_{\pi_t,\pi_{t+1}}) \\ &\quad + H(Z^t_{\sigma_2}|Z^t_{\sigma_3}, Z^t_{\sigma_4}, X_{\pi_t,\pi_{t+1}}) \\ &\overset{(d)}{\le} R^*_{\pi_t,\pi_{t+1}}d + H(Z^t_{\sigma_4}) + H(Z^t_{\sigma_3}|Z^t_{\sigma_4}, D_{\sigma_3}, D_{\sigma_4}) \\ &\quad + H(Z^t_{\sigma_2}|Z^t_{\sigma_3}, Z^t_{\sigma_4}, D_{\sigma_2}, D_{\sigma_3}, D_{\sigma_4}), \end{aligned} \tag{33}$$

where (a) follows from (32), (b) follows from (4) and (8), where $\mathbf{Z}^t_{\sigma[2:4]}$ and $X_{\pi_t,\pi_{t+1}}$ are deterministic functions of the data-set $\mathcal{A}$, (c) from the chain rule of entropy, (d) follows from (30), (31), and because conditioning reduces entropy.

Using the uncoded storage contents in (29), where every worker w_k stores the assigned data point D_k plus separate functions of all the remaining data points, i.e. $D_j(k)$ for $j \in [1:K] \setminus k$, we can write the previous bound as follows

$$\begin{aligned} 4d &\le R^*_{\pi_t,\pi_{t+1}}d + H(Z^t_{\sigma_4}) + H(Z^t_{\sigma_3}|Z^t_{\sigma_4}, D_{\sigma_3}, D_{\sigma_4}) \\ &\quad + H(Z^t_{\sigma_2}|Z^t_{\sigma_3}, Z^t_{\sigma_4}, D_{\sigma_2}, D_{\sigma_3}, D_{\sigma_4}) \\ &\overset{(a)}{=} R^*_{\pi_t,\pi_{t+1}}d + H(D_{\sigma_4}, D_{\sigma_1}(\sigma_4), D_{\sigma_2}(\sigma_4), D_{\sigma_3}(\sigma_4)) \\ &\quad + H(D_{\sigma_1}(\sigma_3), D_{\sigma_2}(\sigma_3)|Z^t_{\sigma_4}) + H(D_{\sigma_1}(\sigma_2)|Z^t_{\sigma_3}, Z^t_{\sigma_4}) \\ &\overset{(b)}{=} R^*_{\pi_t,\pi_{t+1}}d + H(D_{\sigma_4}, D_{\sigma_1}(\sigma_4), D_{\sigma_2}(\sigma_4), D_{\sigma_3}(\sigma_4)) \\ &\quad + H(D_{\sigma_1}(\sigma_3), D_{\sigma_2}(\sigma_3)|D_{\sigma_1}(\sigma_4), D_{\sigma_2}(\sigma_4)) \\ &\qquad + H(D_{\sigma_1}(\sigma_2)|D_{\sigma_1}(\sigma_3), D_{\sigma_1}(\sigma_4)) \\ &= R^*_{\pi_t,\pi_{t+1}}d + H(D_{\sigma_4}) + H(D_{\sigma_3}(\sigma_4)) \\ &\quad + [H(D_{\sigma_1}(\sigma_4)) + H(D_{\sigma_1}(\sigma_3)|D_{\sigma_1}(\sigma_4)) \\ &\qquad + H(D_{\sigma_1}(\sigma_2)|D_{\sigma_1}(\sigma_3), D_{\sigma_1}(\sigma_4))] \\ &\quad + [H(D_{\sigma_2}(\sigma_4)) + H(D_{\sigma_2}(\sigma_3)|D_{\sigma_2}(\sigma_4))] \\ &\overset{(c)}{=} R^*_{\pi_t,\pi_{t+1}}d + d + H(D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4)) \\ &\quad + H(D_{\sigma_2}(\sigma_3, \sigma_4)) + H(D_{\sigma_3}(\sigma_4)) \\ &\overset{(d)}{\le} R^*_{\text{worst-case}}d + d + H(D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4)) \\ &\quad + H(D_{\sigma_2}(\sigma_3, \sigma_4)) + H(D_{\sigma_3}(\sigma_4)), \end{aligned} \tag{34}$$

where (a) follows from the storage content at time t given in (29), where after knowing $\{D_{\sigma_3}, D_{\sigma_4}\}$ (or similarly $\{D_{\sigma_2}, D_{\sigma_3}, D_{\sigma_4}\}$), the only parts left in $Z^t_{\sigma_3}$ (or $Z^t_{\sigma_2}$) are $\{D_{\sigma_1}(\sigma_3), D_{\sigma_2}(\sigma_3)\}$ ($\{D_{\sigma_1}(\sigma_2)\}$), (b) follows since out of the cache contents Z^t_j, the data sub-point $D_k(i)$ only depends on the sub-point $D_k(j)$, for any $i \ne j$, (c) follows from the chain rule of entropy where $D_i(\mathcal{W})$ is the part of D_i stored in the excess storage of all the workers whose indexes are in the set $\mathcal{W}$, and finally (d) follows from Remark 1 where $R^*_{\text{worst-case}} \ge R^*_{\pi_t,\pi_{t+1}}$ for every shuffle (π_t, π_{t+1}).

Summing up over all possible $4! = 24$ permutations of the ordered set $(1, 2, 3, 4)$, we arrive at the following bound,

$$\begin{aligned} &R^*_{\text{worst-case}}d \\ &\ge 3d - \frac{1}{24}\sum_{\sigma\in[4!]}\big[H(D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4)) \\ &\qquad\qquad + H(D_{\sigma_2}(\sigma_3, \sigma_4)) + H(D_{\sigma_3}(\sigma_4))\big] \\ &\overset{(a)}{=} 3d - \frac{1}{24}\sum_{\sigma\in[4!]}\big[H(D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4)) \\ &\qquad\qquad + H(D_{\sigma_1}(\sigma_2, \sigma_3)) + H(D_{\sigma_1}(\sigma_2))\big], \end{aligned} \tag{35}$$

where $[4!]$ is the set of all possible permutations of the ordered set $(1, 2, 3, 4)$, and (a) follows due to the symmetry in the summation by simple change of summation indexes. Following the definition in (14), we can define $D_{k,\mathcal{W}}$ as the part of D_k stored exclusively in the excess storage of the workers whose labels are in the set $\mathcal{W}$. According to $\pi_t = (1, 2, 3, 4)$, at time t, $D_{k,\mathcal{W}}$ is only defined for $k \notin \mathcal{W}$ (w_k does not store D_k as excess storage). Therefore, at time t, we can express the following entropies in terms of $D_{k,\mathcal{W}}$ as follows:

$$\begin{aligned} H(D_k) = \sum_{\mathcal{W}\subseteq[1:4]\setminus k} |D_{k,\mathcal{W}}|d, \quad H(D_k(j)) \\ = \sum_{\substack{\mathcal{W}\subseteq[1:4]\setminus k \\ j\in\mathcal{W}}} |D_{k,\mathcal{W}}|d, \end{aligned} \tag{36}$$

where $|D_{k,\mathcal{W}}|$ is entropy of the sub-point of D_k stored as excess storage only in the set of workers $\mathcal{W}$, i.e., $D_{k,\mathcal{W}}$, normalized by the data point size d. In the summation term of (35), we only have parts of the data points stored in the excess storage of 1, 2, or 3 workers, i.e., $\{D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4), D_{\sigma_1}(\sigma_2, \sigma_3), D_{\sigma_1}(\sigma_2)\}$. Therefore, we obtain the term $|D_{k,\mathcal{W}}|$ only for $|\mathcal{W}| \in \{1, 2, 3\}$. Next, we show how to find the coefficients of $|D_{k,\mathcal{W}}|$ for different sizes of $\mathcal{W}$.

- **Coefficient of** $|D_{k,\mathcal{W}}|$ **for** $|\mathcal{W}| = 1$**:** Due to symmetry, we notice that obtaining the coefficient of $|D_{k,\mathcal{W}}|$ in the summation in (35) for any $|\mathcal{W}| = 1$; is equivalent to obtaining the coefficient of $|D_{1,\{2\}}|$. We get $|D_{1,\{2\}}|$ in the first term of the summation, i.e., $H(D_{\sigma_1}(\sigma_2, \sigma_3, \sigma_4))$ only if $\sigma_1 = 1$ which is satisfied in 6 out of the 24 permutations. In the second term, i.e., $H(D_{\sigma_1}(\sigma_2, \sigma_3))$, we obtain $|D_{1,\{2\}}|$ only if $\sigma_1 = 1$ and $\sigma_4 \ne 2$ in total number of 4 permutations. In the third term, i.e., $H(D_{\sigma_1}(\sigma_2))$, we obtain $|D_{1,\{2\}}|$ only if $\sigma_1 = 1$ and $\sigma_2 = 2$ in total number of 2 permutations. Therefore, the coefficient of $|D_{1,\{2\}}|$, hence any $|D_{k,\mathcal{W}}|$ for $|\mathcal{W}| = 1$, is $\frac{6+4+2}{24} = \frac{1}{2}$.
- **Coefficient of** $|D_{k,\mathcal{W}}|$ **for** $|\mathcal{W}| = 2$**:** Similarly, we obtain the coefficient of $|D_{k,\mathcal{W}}|$ for any $|\mathcal{W}| = 2$ by obtaining the coefficient of $|D_{1,\{2,3\}}|$. We get $|D_{1,\{2,3\}}|$ in the first two terms

of the summation only if $\sigma_1 = 1$ which is satisfied in 6 out of the 24 permutations. In the third term, we obtain $|D_{1,\{2,3\}}|$ only if $\sigma_1 = 1$ and $\sigma_2 \in \{2, 3\}$ in total number of 4 permutations. Therefore, the coefficient of $|D_{1,\{2,3\}}|$, hence any $|D_{k,\mathcal{W}}|$ for $|\mathcal{W}| = 2$, is $\frac{6+6+4}{24} = \frac{2}{3}$.

- **Coefficient of $|D_{k,\mathcal{W}}|$ for $|\mathcal{W}| = 3$:** We obtain the coefficient of $|D_{k,\mathcal{W}}|$ for any $|\mathcal{W}| = 3$ by obtaining the coefficient of $|D_{1,\{2,3,4\}}|$. We get $|D_{1,\{2,3,4\}}|$ in the first three terms of the summation only if $\sigma_1 = 1$ which is satisfied in 6 out of the 24 permutations. Therefore, the coefficient of $|D_{1,\{2,3,4\}}|$, hence any $|D_{k,\mathcal{W}}|$ for $|\mathcal{W}| = 3$, is $\frac{6+6+6}{24} = \frac{3}{4}$.

Therefore, we can simplify the bound in (35) as follows:

$$
\begin{aligned}
R^*_{\text{worst-case}} d &\geq 3d - \frac{1}{2}\sum_{k=1}^{4} \sum_{\substack{\mathcal{W}\subseteq[1:K]\backslash k\\ |\mathcal{W}|=1}} |D_{k,\mathcal{W}}| d \\
&\quad - \frac{2}{3}\sum_{i=1}^{4} \sum_{\substack{\mathcal{W}\subseteq[1:K]\backslash k\\ |\mathcal{W}|=2}} |D_{k,\mathcal{W}}| d - \frac{3}{4}\sum_{k=1}^{4} \sum_{\substack{\mathcal{W}\subseteq[1:K]\backslash k\\ |\mathcal{W}|=3}} |D_{k,\mathcal{W}}| d \\
&= 3d - \frac{x_1 d}{2} - \frac{2x_2 d}{3} - \frac{3x_3 d}{4}, \qquad (37)
\end{aligned}
$$

where x_ℓ for $\ell \in [0:3]$ is defined similar to (16) as $x_\ell = \sum_{k=1}^{K} \sum_{\mathcal{W}\subseteq[1:4]\backslash k:\, |\mathcal{W}|=\ell} |D_{k,\mathcal{W}}|$. By dividing both sides by d, we get the following bound:

$$
R^*_{\text{worst-case}} \geq 3 - \frac{x_1}{2} - \frac{2x_2}{3} - \frac{3x_3}{4}. \qquad (38)
$$

Moreover, the data size and the excess storage size constraints for this example follow (15) and (17), respectively. Hence, we obtain the following constraints:

$$
\begin{aligned}
x_0 + x_1 + x_2 + x_3 &= 4, \qquad (39)\\
x_1 + 2x_2 + 3x_3 &\leq 4(S-1). \qquad (40)
\end{aligned}
$$

We get the first bound on $R^*_{\text{worst-case}}$ by eliminating x_1 from (38) using the bound in (40) as follows:

$$
\begin{aligned}
R^*_{\text{worst-case}} &\geq 3 - \frac{x_1}{2} - \frac{2x_2}{3} - \frac{3x_3}{4} \\
&\geq 3 - \frac{1}{2}\left(4(S-1) - 2x_2 - 3x_3\right) - \frac{2x_2}{3} - \frac{3x_3}{4} \\
&= 5 - 2S + \frac{x_2}{3} + \frac{3x_3}{4} \overset{(a)}{\geq} 5 - 2S, \qquad (41)
\end{aligned}
$$

where (a) follows since $x_2, x_3 \geq 0$.

We get the second bound on $R^*_{\text{worst-case}}$ in two steps. First, we eliminate x_1 from (38) and (40) using (39) to get the following two bounds:

$$
\begin{aligned}
R^*_{\text{worst-case}} &\geq 3 - \frac{1}{2}\left(4 - x_0 - x_2 - x_3\right) - \frac{2x_2}{3} - \frac{3x_3}{4} \\
&= 1 + \frac{x_0}{2} - \frac{x_2}{6} - \frac{x_3}{4}, \qquad (42)\\
4(S-1) &\geq (4 - x_0 - x_2 - x_3) + 2x_2 + 3x_3 \\
&= 4 - x_0 + x_2 + 2x_3. \qquad (43)
\end{aligned}
$$

We eliminate x_2 from (42) using the bound in (43) to obtain

$$
\begin{aligned}
R^*_{\text{worst-case}} &\geq 1 + \frac{x_0}{2} - \frac{x_2}{6} - \frac{x_3}{4} \\
&\geq 1 + \frac{x_0}{2} - \frac{1}{6}\left(4(S-1) - 4 + x_0 - 2x_3\right) - \frac{x_3}{4} \\
&= \frac{7}{3} - \frac{2S}{3} + \frac{x_0}{3} + \frac{x_3}{12} \overset{(a)}{\geq} \frac{7-2S}{3}, \qquad (44)
\end{aligned}
$$

where (a) follows since $x_0, x_3 \geq 0$.

Following similar steps, we get a third bound on $R^*_{\text{worst-case}}$ by first eliminating x_2 from (38) and (40) using (39) to get the following two bounds:

$$
\begin{aligned}
R^*_{\text{worst-case}} &\geq 3 - \frac{x_1}{2} - \frac{2}{3}\left(4 - x_0 - x_1 - x_3\right) - \frac{3x_3}{4} \\
&= \frac{1}{3} + \frac{2x_0}{3} + \frac{x_1}{6} - \frac{x_3}{12}, \qquad (45)\\
4(S-1) &\geq x_1 + 2\left(4 - x_0 - x_1 - x_3\right) + 3x_3 \\
&= 8 - 2x_0 - x_+ x_3. \qquad (46)
\end{aligned}
$$

We eliminate x_3 from (45) using the bound in (46) and arrive to

$$
\begin{aligned}
R^*_{\text{worst-case}} &\geq \frac{1}{3} + \frac{2x_0}{3} + \frac{x_1}{6} - \frac{x_3}{12} \geq \frac{1}{3} + \frac{2x_0}{3} + \frac{x_1}{6} \\
&\quad - \frac{1}{12}\left(4(S-1) - 8 + 2x_0 + x_1\right) \\
&= \frac{4}{3} - \frac{S}{3} + \frac{5x_0}{6} + \frac{x_1}{12} \overset{(a)}{\geq} \frac{4-S}{3}, \qquad (47)
\end{aligned}
$$

where (a) follows since $x_0, x_1 \geq 0$.

In summary, we obtain in (41), (44), and (47) the following bounds on $R^*_{\text{worst-case}}$:

$$
R^*_{\text{worst-case}} \geq 5 - 2S, \quad R^*_{\text{worst-case}} \geq \frac{7-2S}{3}, \quad R^*_{\text{worst-case}} \geq \frac{4-S}{3}. \qquad (48)
$$

The intersection of the three bounds is the lower convex hull of the 4 storage-rate pairs, ($S = m$, $R = \frac{4-m}{m}$) for $m \in [1:4]$, which is the obtained lower bound on $R^*_{\text{worst-case}}$ given by the blue curve in Figure 1, satisfying Theorem 2 for $K = N = 4$.

C. Theorem 3: Gap Between the Bounds in Theorems 1 and 2

In our next result, we compare the upper and lower bounds in Theorems 1 and 2, respectively, and show that they are within a constant multiplication gap of each other.

Theorem 3: For a data-set containing $N \in \mathbb{N}$ data points, and a set of $K \in \mathbb{N}$ distributed workers, the gap ratio between the upper and the lower bounds on $R^*_{\text{worst-case}}$ given by Theorems 1, and 2, respectively, is bounded as follows:

$$
\frac{R^{\text{upper}}_{\text{worst-case}}}{R^{\text{lower}}_{\text{worst-case}}} \leq \frac{K}{K-1} \leq 2. \qquad (49)
$$

The formal proof for the maximum gap analysis for any value of K and N can be found in Appendix III. This Theorem shows that there is a vanishing gap between the bounds as the number of workers K increases, i.e., $\lim_{K\to\infty}\left(\frac{K}{K-1}\right) = 1$. We also show that for the discrete set of storage points considered in Theorem 1, i.e., $S = \left(1 + i\,\frac{K-1}{K}\right)\frac{N}{K}$ for $i \in [1:K]$, our achievable scheme is in fact optimal, and that the gap only results in the values of storage in between, i.e., memory sharing is not optimal in this case. For example, consider the bounds on $R^*_{\text{worst-case}}$ for $K = N = 4$ shown in Figure 1. We first notice that the achieved storage-rate pairs $(S = 7/4, R = 3/2)$, $(S = 5/2, R = 2/3)$, and $(S = 13/4, R = 1/4)$ are optimal. Furthermore, we can show that the maximum gap between the bounds is at $S = 1$, which is given by $4/3$, satisfying the bound in (49).

D. Theorem 4: Improved Gap Between the Bounds in Theorems 1 and 2

The next Theorem provides an improved gap through a new achievable scheme, which we call as *"aligned coded shuffling"*.

Theorem 4: For a data-set containing $N \in \mathbb{N}$ data points, and a set of $K \in \mathbb{N}$ distributed workers, the lower bound on $R^*_{\text{worst-case}}$ in Theorem 2 is in fact achievable for $K < 5$ (hence gives the optimal rate), while for $K \geq 5$ is achievable within a gap ratio bounded as

$$\frac{R^{\text{upper}}_{\text{worst-case}}}{R^{\text{lower}}_{\text{worst-case}}} \leq \frac{K - \frac{1}{3}}{K - 1} \leq \frac{7}{6}. \tag{50}$$

In Appendix IV, we present the complete proof of Theorem 4. The above theorem is proved by closing the gap between the two bounds in Theorems 1 and 2 for the storage values $S = m\frac{N}{K}$, and $m \in \{1, K-2, K-1\}$. This can be done with the use of sophisticated interference alignment mechanisms, which force the interference seen by each worker to occupy the minimum possible dimensions. To illustrate the new ideas introduced here, we revisit again Example III-A.1 of $K = N = 4$ to show how the gap between the lower and the upper bounds on $R^*_{\text{worst-case}}$ can be closed in this case.

1) Example 3: Optimal Worst-Case Rate for $N = 4$ and $K = 4$: From Figure 1, we notice that if we close the gap for the storage points $S = m$, for $m \in [1 : 3]$, then we can fully characterize $R^*_{\text{worst-case}}$ using memory sharing between the achievable points (see Claim 1). In the achievability, we consider a different placement strategy, which is also invariant in the structure. We also consider the aligned coded shuffling scheme for data delivery, which reduces the rate by forcing the interference to occupy the minimum possible dimensions. We consider the same subsequent shuffles $\pi_t = (1, 2, 3, 4)$, and $\pi_{t+1} = (2, 3, 4, 1)$. Furthermore, we define $\delta_t(i)$ as the index of the worker being assigned the data point D_i at time t. Therefore, $\delta_t = (1, 2, 3, 4)$, and $\delta_{t+1} = (4, 1, 2, 3)$. Next, we discuss the achievability for storage values $S = m$, and $m \in [1 : 3]$.

- **Case $\mathbf{m = 1}$ ($\mathbf{S = 1}$):**

As mentioned before in Example III-A.1, the storage placement for the case $m = 1$ (no excess storage) is trivial where every worker only stores the data point which needs to be processed. We start by sending 3 independent linear combinations of the 4 data points as follows:

$$X_{\pi_t,\pi_{t+1}} = \left\{ \begin{array}{l} L_1(D_1, D_2, D_3, D_4), \\ L_2(D_1, D_2, D_3, D_4), \\ L_3(D_1, D_2, D_3, D_4) \end{array} \right\}. \tag{51}$$

where L_1, L_2, and L_3 are three independent linear functions. We notice that each worker already stores one data point, and then can decode the 3 remaining data points and acquire the one needed at time $t+1$. For instance, worker w_1 has D_1 from the previous shuffle at time t, and then can get 3 independent linear functions enough to decode D_2, D_3, and D_4. Therefore, the pair $(S = 1, R = 3)$ is achievable for $K = N = 4$ closing the gap in Figure 1 for $S = 1$. The storage update is also trivial, where every worker keeps the new assigned data point and discard the remaining three points.

- **Case $\mathbf{m = 2}$ ($\mathbf{S = 2}$):**

Storage Placement: Every data point at time t is partitioned into 3 sub-points of size $d/3$ bits each, where every sub-point of the data point D_i is labeled by a unique subset $\mathcal{W} \subseteq [1 : 4] \setminus \delta_t(i)$, where $|\mathcal{W}| = 1$. For example, the data point D_1 at time t is partitioned as $D_1 = \{D_{1,\{2\}}, D_{1,\{3\}}, D_{1,\{4\}}\}$. The storage placement at time t follows from Figure 5a. First, every worker stores the data point needed to be processed. Then, in the excess storage, every worker w_k stores the sub-points labeled by the set $\mathcal{W}$, where $k \in \mathcal{W}$, e.g., w_1 stores the sub-point $\mathcal{A}^t(2, 1) = \{D_{2,\{1\}}\}$ from D_2. To summarize, each worker stores the assigned data point of size d bits, and for each one of the remaining 3 data points, it stores 1 sub-point of size $d/3$ bits. That is, the storage requirement is given by $S = 1 + 3 \times 1 \times 1/3 = 2$, which satisfies the storage constraint.

Aligned Coded Shuffling: According to the storage placement at time t in Figure 5a, at time $t+1$ every worker needs 2 sub-points of the assigned data point, where every needed sub-point is available at exactly 2 other workers. From an interference perspective, every one of the needed sub-points is an interference to only one worker, e.g., $D_{3,\{4\}}$ needed by w_2 at time $t+1$, is available at w_3 and w_4, and cause interference at w_1 (neither needed nor available). Therefore, w_1 can face interference from total 2 sub-points: $D_{3,\{4\}}$ (needed by w_2), and $D_{4,\{2\}}$ (needed by w_3). By aligning these two sub-points and considering the coded symbol $D_{3,\{4\}} \oplus D_{4,\{2\}}$, we notice the following: 1) This coded symbol is available at the worker w_4; 2) It is useful for the two workers w_2, and w_3 at the same time; and 3) It is the only source of interference for w_1. Similarly, we can produce 3 more aligned symbols to get in total 4 aligned coded symbols, of size $d/3$ bits each, summarized as follows:

$$\begin{aligned} D_{3,\{4\}} \oplus D_{4,\{2\}} &: \text{Interference at } w_1, \text{ available at } w_4, \\ &\qquad \text{and useful for } w_2, \text{ and } w_3; \\ D_{1,\{3\}} \oplus D_{4,\{1\}} &: \text{Interference at } w_2, \text{ available at } w_1, \\ &\qquad \text{and useful for } w_3, \text{ and } w_4; \\ D_{1,\{2\}} \oplus D_{2,\{4\}} &: \text{Interference at } w_3, \text{ available at } w_2, \\ &\qquad \text{and useful for } w_1, \text{ and } w_4; \\ D_{2,\{3\}} \oplus D_{3,\{1\}} &: \text{Interference at } w_4, \text{ available at } w_3, \\ &\qquad \text{and useful for } w_1, \text{ and } w_2. \end{aligned} \tag{52}$$

Therefore, these 4 coded symbols provide every worker with the 2 needed sub-points. Moreover, it suffices to send only three independent linear combinations of these 4 coded symbols as shown in Figure 5b, since every worker already has one of them available locally at its storage. The rate of this

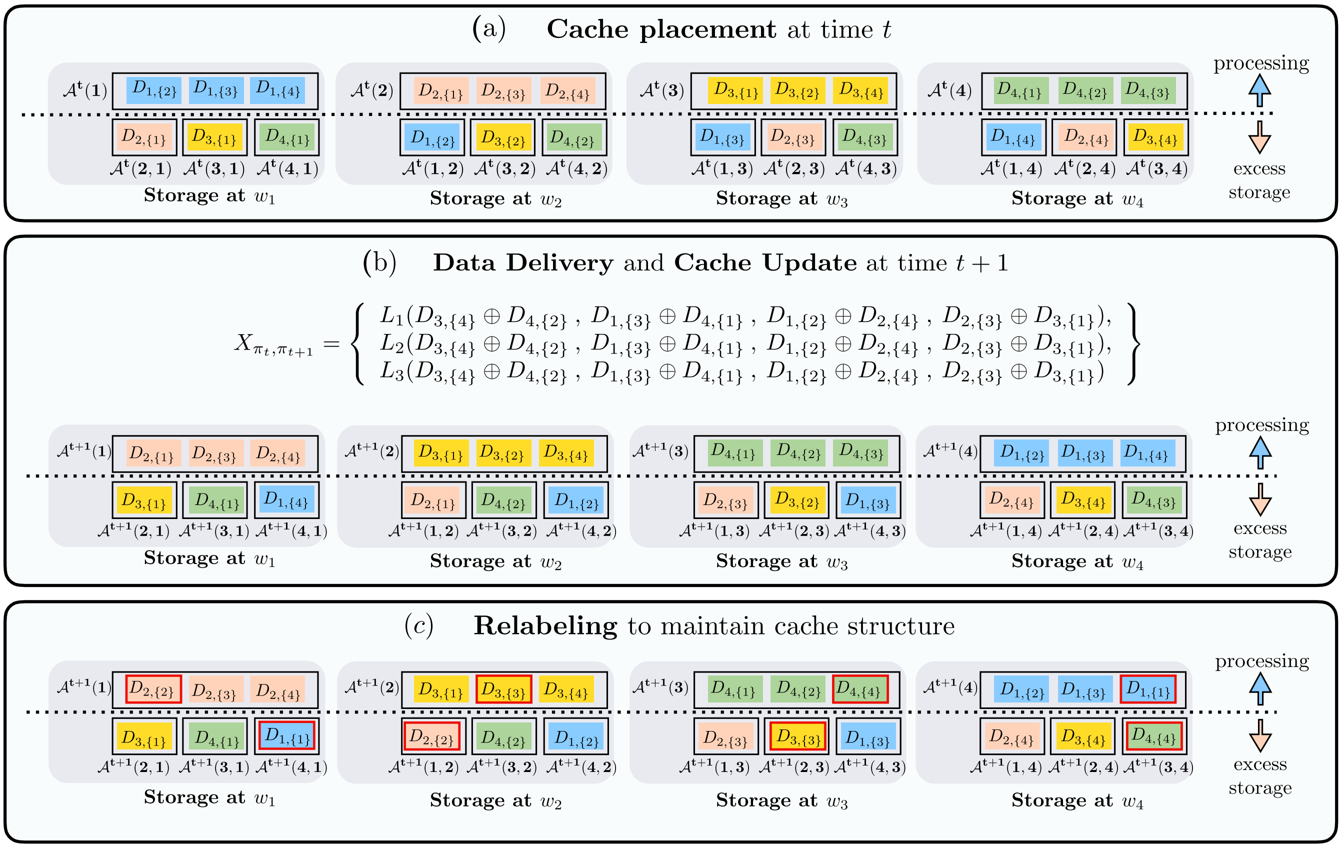

Fig. 5. An example on closing the gap of $K = 4$ workers, $N = 4$ data points, and $m = 2$ ($S = 2$): (a) New SIP mechanism, (b) Data Delivery and storage update, and (c) Relabeling some sub-points in red frames to maintain the storage structure. At time t, every data point D_i is partitioned into 3 sub-points each labeled by a unique subset of length 1 of the set $[1:4] \setminus \delta_t(i)$. At time $t+1$, for every data point D_i the sub-point $D_{i,\{\delta_{t+1}(i)\}}$ is relabeled as $D_{i,\{\delta_t(i)\}}$. Above the dotted line is the data point fully stored for processing, and below the dotted line is the excess storage used to store the sub-points containing the worker's index.

transmission is $R = 3 \times 1/3 = 1$, and the pair ($S = 2$, $R = 1$) is achievable which closes the gap in Figure 1 for $S = 2$.

Storage Update and Sub-points Relabeling: The storage update at time $t+1$ is done in a way that preserves the structure of the storage at time t. The storage update is shown in Figure 5b, while the relabeling process is shown in Figure 5c for the sub-points in red frames. A data point D_i is processed by the workers $w_{\delta_t(i)}$, and $w_{\delta_{t+1}(i)}$ at epochs t, and $t+1$, respectively. At epoch t, the worker $w_{\delta_t(i)}$ already has D_i completely, and therefore it is not stored in its excess storage. In the same time epoch t, the worker $w_{\delta_{t+1}(i)}$ has only part of D_i stored in its excess storage, i.e., $\mathcal{A}^t(\delta_t(i), \delta_{t+1}(i)) = \{D_{i,\{\delta_{t+1}(i)\}}\}$. At epoch $t+1$, the worker $w_{\delta_{t+1}(i)}$ now gets D_i completely in its storage. In order to maintain the global structure of the storage, the worker $w_{\delta_t(i)}$ will only keep the part of D_i in its excess storage at time $t+1$ (stored completely at time t) which was stored within the excess storage of the worker $w_{\delta_{t+1}(i)}$ at time t, i.e., $\mathcal{A}^{t+1}(\delta_{t+1}(i), \delta_t(i)) = \mathcal{A}^t(\delta_t(i), \delta_{t+1}(i)) = \{D_{i,\{\delta_{t+1}(i)\}}\}$. Since the data sub-point $D_{i,\{\delta_{t+1}(i)\}}$ is stored in the excess storage of the worker $w_{\delta_t(i)}$ at time $t+1$, it needs to be relabeled as $D_{i,\{\delta_t(i)\}}$, i.e.,

$$\begin{aligned} A^t(\delta_t(i), \delta_{t+1}(i)) &= \{D_{i,\{\delta_{t+1}(i)\}} \longrightarrow D_{i,\{\delta_t(i)\}}\} \\ &= A^{t+1}(\delta_{t+1}(i), \delta_t(i)), \quad \forall i \in [1:4]. \end{aligned} \tag{53}$$

For example as shown in Figure 5b, D_1 is assigned to workers w_1, and w_4 at time epochs t, and $t+1$, respectively. Therefore at time $t+1$, w_1 will keep the sub-point $A^{t+1}(4,1) = A^t(1,4) = \{D_{1,\{4\}}\}$ of D_1 in its excess storage. As shown in Figure 5c, $D_{1,\{4\}}$ is relabeled as $D_{1,\{1\}}$ since it is now stored in the excess storage of worker w_1:

$$A^t(1,1) = \{D_{1,\{4\}} \longrightarrow D_{1,\{1\}}\} = A^{t+1}(4,1), \tag{54}$$

which preserves the structure of the storage.

- **Case m = 3 (S = 3)**:

Storage Placement: Every data point at time t is partitioned into 3 sub-points of size $d/3$ bits each, where every sub-point of the data point D_i is labeled by a unique subset $\mathcal{W} \subseteq [1:4] \setminus \delta_t(i)$, where $|\mathcal{W}| = 2$. For example, the data point D_1 at time t is partitioned as $D_1 = \{D_{1,\{2,3\}}, D_{1,\{2,4\}}, D_{1,\{3,4\}}\}$. The storage placement at time t follows from Figure 6a. First, every worker stores the data point needed to be processed. Then, in the excess storage, every worker w_k stores the sub-points labeled by the set $\mathcal{W}$, where $k \in \mathcal{W}$, e.g., w_1 stores the two sub-points $\mathcal{A}^t(2,1) = \{D_{2,\{1,3\}}, D_{1,\{1,4\}}\}$ from D_2. To summarize, each worker stores the assigned data point of size d bits, and for each one of the remaining 3 data points, it stores 2 sub-points of size $d/3$ bits each. That is, the storage requirement is given by $S = 1 + 3 \times 2 \times 1/3 = 3$, which satisfies the storage constraint.

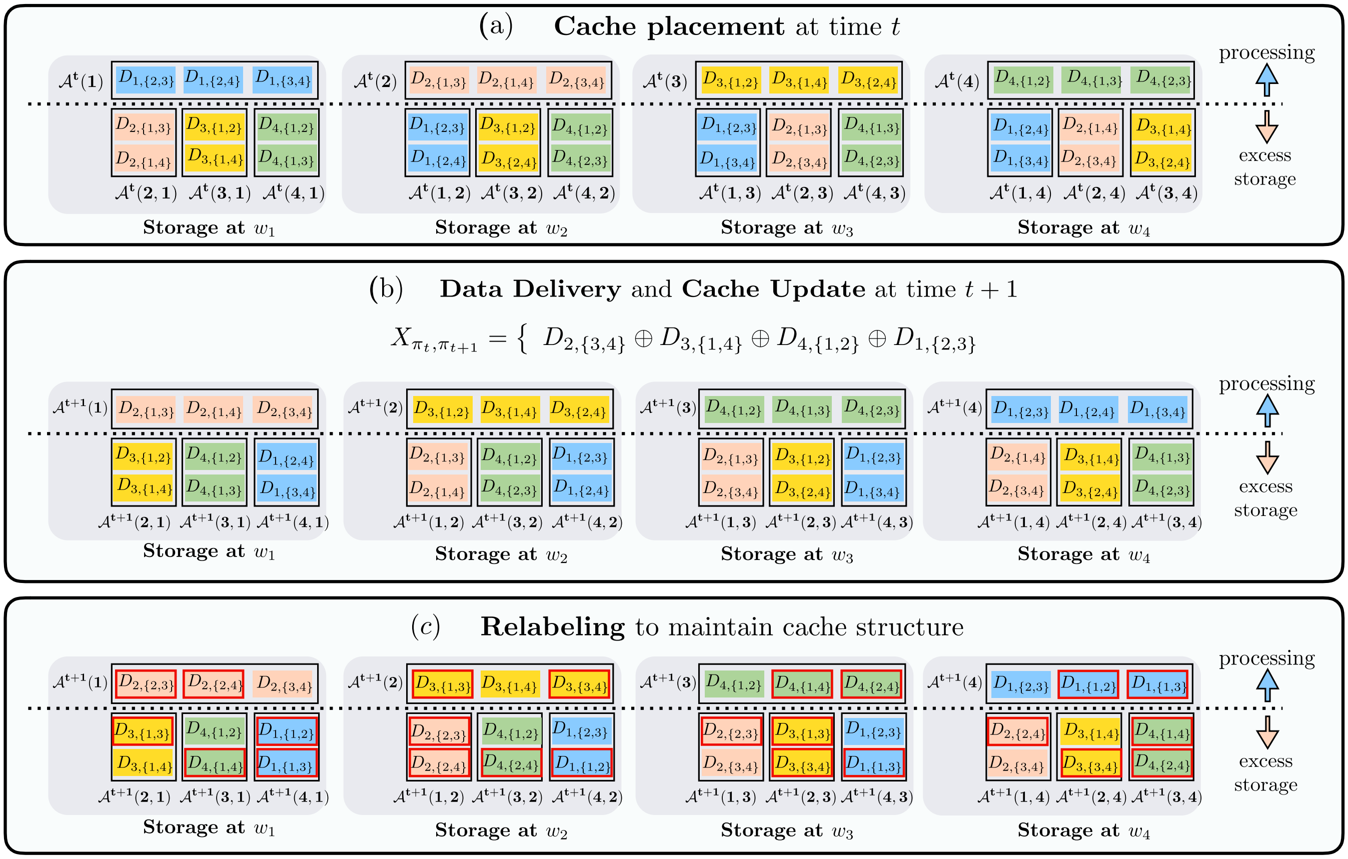

Fig. 6. An example on closing the gap of $K = 4$ workers, $N = 4$ data points, and $m = 3$ ($S = 3$): (a) new SIP mechanism, (b) Data Delivery and storage update, and (c) Relabeling some sub-points in red frames to maintain the storage structure. At time t, every data point D_i is partitioned into 3 sub-points each labeled by a unique subset of length 2 of the set $[1:4] \setminus \delta_t(i)$. At time $t+1$, for every data point D_i the sub-point $D_{i,\mathcal{W}}$ where $\delta_{t+1}(i) \in \mathcal{W}$ is relabeled by replacing $\delta_{t+1}(i)$ with $\delta_t(i)$. Above the dotted line is the data point fully stored for processing, and below the dotted line is the excess storage used to store the sub-points containing the worker's index.

Aligned Coded Shuffling: According to the storage placement at time t in Figure 6a, at time $t+1$ every worker needs only one sub-point of the assigned data point, which is available at the 3 other workers, e.g., w_1 needs $D_{2,\{3,4\}}$ which is available at the workers w_2, w_3, and w_4. Therefore, we can send an order 4 symbol useful for the 4 workers at the same time as follows:

$$X_{\pi_t,\pi_{t+1}} = \left\{D_{2,\{3,4\}} \oplus D_{3,\{1,4\}} \oplus D_{4,\{1,2\}} \oplus D_{1,\{2,3\}}\right\}. \quad (55)$$

The rate of this transmission is $R = 1 \times 1/3 = 1/3$, and the pair $(S = 3, R = 1/3)$ is achievable which closes the gap in Figure 1 for $S = 3$.

Storage Update and Sub-points Relabeling: Similar to the case $m = 2$, the storage update for the case $m = 3$ is shown in Figure 6b, and the relabeling process is shown in Figure 6c for the sub-points in red frames. For every data point D_i, the worker $w_{\delta_t(i)}$, which already has D_i completely at time t, will only keep at time $t+1$ the part of D_i stored within the excess storage of the worker $w_{\delta_{t+1}(i)}$ at time t, i.e., $\mathcal{A}^{t+1}(\delta_{t+1}(i), \delta_t(i)) = \mathcal{A}^t(\delta_t(i), \delta_{t+1}(i)) = \{D_{i,\mathcal{W}}\}$, where $\delta_{t+1}(i) \in \mathcal{W}$. Then, we relabel this set of sub-points by replacing $\delta_{t+1}(i)$ in $\mathcal{W}$ with $\delta_t(i)$, i.e.,

$$\begin{aligned} A^t(\delta_t(i),\delta_{t+1}(i)) = \big\{D_{i,\mathcal{W}} : \delta_{t+1}(i) \in \mathcal{W} \longrightarrow \\ D_{i,\mathcal{W}'} : \mathcal{W}' = \mathcal{W} \cup \{\delta_t(i)\} \setminus \{\delta_{t+1}(i)\}\big\} \\ = A^{t+1}(\delta_{t+1}(i), \delta_t(i)), \quad \forall i \in [1:4]. \end{aligned} \quad (56)$$

For example, the data point D_1 is processed by w_1, and w_4 in the epochs t, and $t+1$, respectively. As shown in Figure 6b, w_1 will keep the sub-points $A^{t+1}(4,1) = A^t(1,4) = \{D_{1,\{2,4\}}, D_{1,\{3,4\}}\}$ of D_1 in its excess storage at time $t+1$. Then, the following relabeling is done to the sub-points of D_1 as shown in Figure 6c:

$$\begin{aligned} A^t(1,1) = \{D_{1,\{2,4\}} \longrightarrow D_{1,\{1,2\}}, \ \textbf{and} \\ D_{1,\{3,4\}} \longrightarrow D_{1,\{1,3\}}\} = A^{t+1}(4,1), \end{aligned} \quad (57)$$

which preserves the structure of the storage.

As a conclusion for the example $K = N = 4$, the lower convex envelope of the achievable pairs $(S = m, R = \frac{4-m}{m})$, for $m \in [1:4]$, is the optimal storage-rate trade-off.

IV. Comparisons and Simulations

In this Section, we conduct some numerical simulations to compare the *average performance* of our proposed coded shuffling scheme using SIP with the probabilistic scheme using random placement in [20]. While the Aligned Coded Shuffling scheme is proposed in this paper to achieve the optimal worst-case rate for some storage values, we do not use it in our simulations since it is not generalized for any number of workers K.

In our first set of simulations, we plot the normalized shuffling rate in Figure 7 for probabilistic coding scheme using

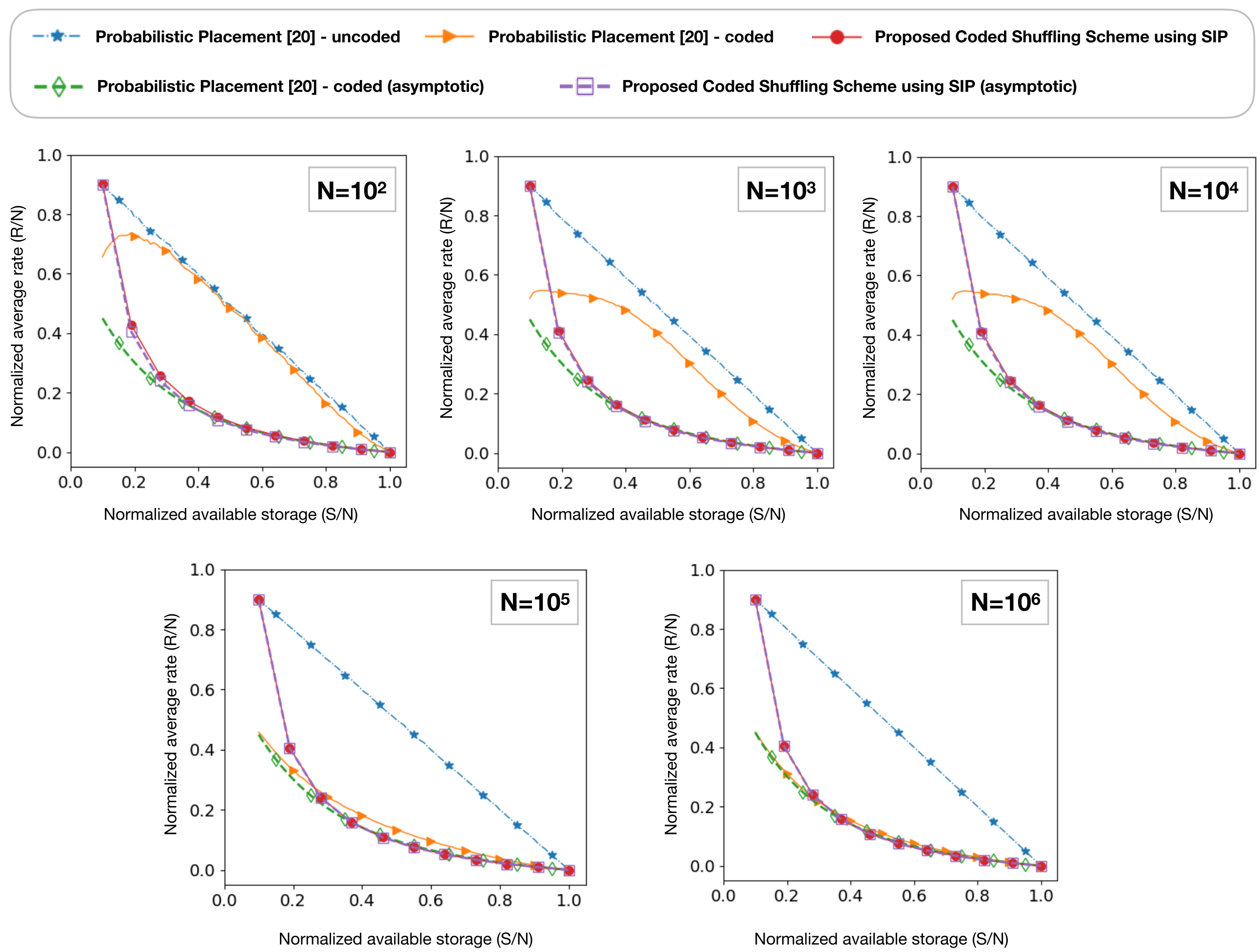

Fig. 7. Average normalized rate for probabilistic coding scheme using random placement versus our proposed scheme using SIP for five different values of number of data points $N \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ and $K = 10$ workers.

random placement versus our proposed scheme using SIP, averaged over 10^3 random shuffles. We consider five different values of number of data points $N \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ and $K = 10$ workers. For the random placement, we consider both the uncoded and the probabilistic coded schemes. We also plot the theoretical guarantees on the rate for our proposed scheme given by (63), as well as the probabilistic coded shuffling scheme given in [20] as follows,

$$\lim_{N\to\infty} R_{\text{Coded-Prob}} = \frac{N}{(pK)^2}\left((1-p)^{K+1} + (K-1)p(1-p) - (1-p)^2\right), \quad (58)$$

where $p = \frac{S-\frac{N}{K}}{N-\frac{N}{K}}$.

In Figure 7, we notice that generally our proposed scheme achieves average rates very close to the theoretical guarantees (asymptotic) in (63) even for small values of N. However, the probabilistic coded scheme is far from its theoretical guarantees (58) for small values of N and gets closer as the number of data points N increases, an observation that was first addressed and studied in [4]. We can also notice that except for small storage values, our proposed scheme outperforms the probabilistic scheme for different values of N.

We also note that it is possible to further improve the proposed coded shuffling scheme in the small storage regime, since it does not leverage all possible coding opportunities provided by the structural invariant placement strategy (SIP). For instance, our scheme only sends uncoded symbols for the no-excess storage case, while order-2 symbols can still be used as discussed in Appendix IV-B. In fact, the recent subsequent work of Elmahdy and Mohajer [40] precisely exploits such opportunities to further reduce the rate.

Furthermore, we show the power of our novel modified SIP mechanism (in Appendix IV-A) compared to the random placement in [20]. In our second simulation, we compare the recent coded shuffling scheme proposed in [40] based on the modified SIP mechanism for the special case when the number of workers equals to the number of data-points, i.e., $N = K$, since this scheme is not generalized yet for all values of N and K and any arbitrary shuffle. The simulations results are shown in Figure 8 for $N = K = 10$. We notice that the new scheme in [40] outperforms the probabilistic scheme in [20] for any value of storage. In general, the scheme in [40] is proven optimal for the special $N = K$ and any arbitrary shuffle.

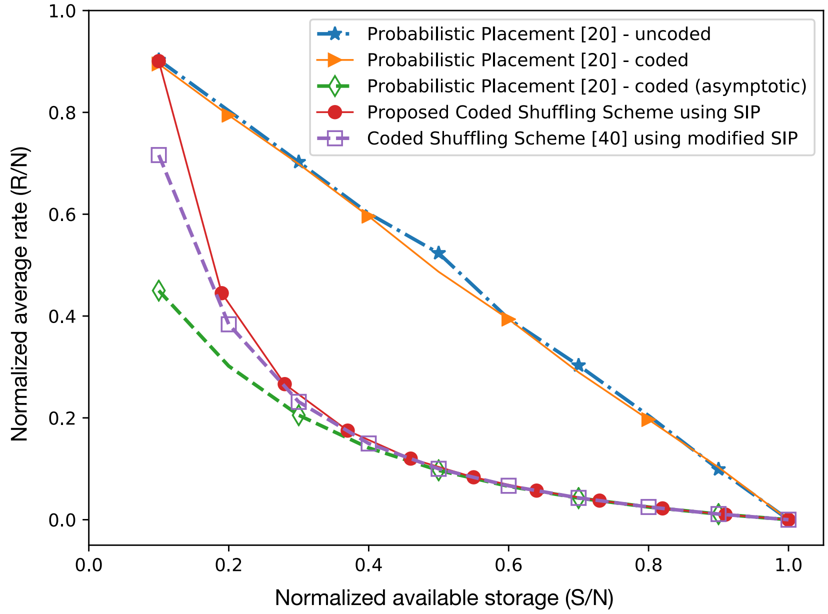

Fig. 8. Average rate for probabilistic scheme using random placement, our proposed coded shuffling scheme using modified SIP, and the new coded shuffling scheme in [40] using SIP for $N = 10$ data points, and $K = 10$ workers.

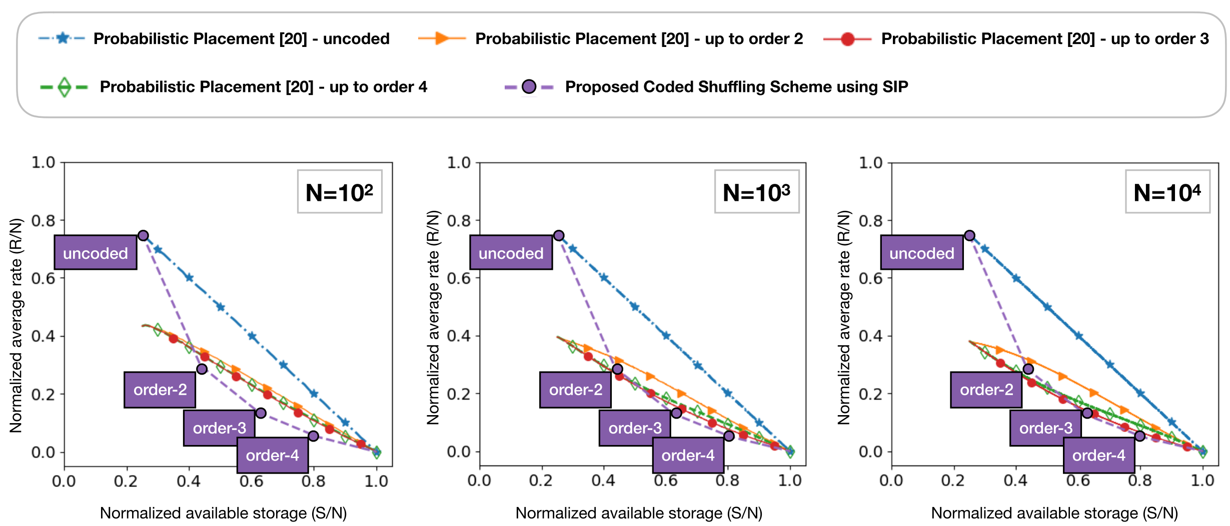

Fig. 9. Average rate for probabilistic scheme using random placement versus our proposed scheme using SIP for $N =\in \{10^2, 10^3, 10^4\}$ data points, and $K = 4$ workers. For probabilistic scheme, we consider the following coding opportunities: a) uncoded transmission; b) up to order-2 coded symbols; c) up to order-3 coded symbols; and d) up to order-4 coded symbols.

Since the asymptotic rate of the probabilistic scheme is lower than the optimal trade-off given by the coded shuffling scheme in [40], this means that (58) is not necessary achievable for small values of N and only gives the theoretical guarantees for for large enough N.

It is noteworthy here that while the scheme in [20] requires exhaustive search of all coding opportunities, i.e., coded symbols of order i for $i \in [2 : K]$, our scheme only uses for each value of storage coded symbols of the same order, i.e., for the storage values $S = \left(1 + i\left(\frac{K-}{K}\right)\right)\frac{N}{K}$ where $i \in [0 : K]$, only order i symbols are transmitted. Therefore, our scheme has significantly reduced encoding and decoding complexity compared to [20].

In order to study the complexity of the probabilistic storage placement scheme in [20] versus our proposed scheme, we present simulation results in Figure 9 for three different values of number of data points $N \in \{10^2, 10^3, 10^4\}$, and $K = 4$ workers. For $K = 4$, we only have coding opportunities up to order-4 symbols using random placement. Therefore, in our simulations, we consider the rate of the probabilistic scheme by leveraging the following coding opportunities: a) uncoded transmission; b) up to order-2 coded symbols; c) up to order-3 coded symbols; and d) up to order-4 coded symbols. Figure 9 shows that our scheme generally outperforms the probabilistic scheme with the corresponding level of coding leveraging for different values of N, e.g., the rate of our proposed scheme when order-2 symbols is used is lower than the rate of the probabilistic scheme when up to order-2 symbols are used, i.e., lower rate for same coding complexity. Moreover, our scheme outperforms the probabilistic scheme even when all coding opportunities are leveraged for large values of storage. We notice that while our proposed scheme behaves similarly for different N values, the probabilistic coded scheme rate decreases with respect to the rate of our scheme as the value of N grows larger since it approaches the theoretical asymptotic guarantees as given in (58).

V. Conclusion

We considered the worst-case trade-off between the amount of storage and communication overhead for the data shuffling problem. First, we presented an information theoretic formulation of the problem. Following that, we proposed a novel uncoded-structural invariant storage placement and update (SIP/SIU) strategy for different storage values at the workers. This placement strategy allowed for applying a similar coding scheme to the one in [43]. Through a novel bounding methodology similar to [41], [42], we derived an information theoretic lower bound on the worst-case communication rate as a function of the storage, which showed that the resulting communication overhead of our scheme is within a maximum multiplicative factor of $\frac{K}{K-1}$, where K is the number of workers. Furthermore, we presented a new scheme inspired by the idea of interference alignment, which closes the gap and hence achieves the optimal worst-case rate-storage trade-off for $K < 5$, and further reduces the maximum multiplicative factor to $\frac{K-\frac{1}{3}}{K-1}$ for $K \geq 5$.

While our SIP strategy provides coding opportunities and is proven optimal in the worst-case shuffle for some values of K, it requires arbitrary large number of divisions for each data point. For instance, in the aligned coded shuffling scheme we need $\binom{K-1}{m-1}$ number of data point divisions for the storage value $S = m\frac{N}{K}$. Adding data divisibility constraints has been considered recently in the caching literature [45]–[48] to restrict finite number of files divisions, while achieving comparable levels of coding gain. An interesting future direction is to adapt some of these approaches to the problem of distributed data shuffling.

Appendix I
Upper Bound on $R^*_{\text{worst-case}}$ (Proof of Theorem 1)

Following Example III-A.1, we present our general achievability for any number of workers K, any number of data points N, and any storage value S. Our scheme has two main phases: structural invariant storage placement/update phase (SIP/SIU); and data delivery phase. This scheme also proves the upper bound on the optimal worst-case rate, i.e., $R^{\text{upper}}_{\text{worst-case}}$ stated in Theorem 1.

A. Structural Invariant Placement (SIP)

We first propose the SIP mechanism, which allows applying a similar data delivery scheme to the one proposed in [43]. The placement procedure involves updating the storage content for each worker after each shuffle in order to maintain the structure of the storage. Since the shuffling process at each time is done randomly, all the data points not being processed by a worker w_k are of equal importance to reduce the communication overhead in the next shuffle. Consequently, the amount of excess storage of size $\left(S - \frac{N}{K}\right) d$ bits is equally divided among these points, where we assume uncoded storage placement.

We focus on a discrete set of storage values given by $S = \left(1 + i(\frac{K-1}{K})\right) \frac{N}{K}$, for $i \in [0 : K]$. The values in between can then be achieved by memory sharing as stated in Claim 1. At time t, the worker w_k first stores the batch assigned for processing, $\mathcal{A}^t(k)$, in order to satisfy the processing constraint in (6), which requires $\frac{N}{K}d$ bits of the available storage. That is if a data point $D \in \mathcal{A}^t(k)$, then D is fully stored in Z_k^t. Assume in the following that the dimensionality of the data points d is integer multiples of $\binom{K}{i}$. The excess storage of size $(S-\frac{N}{K})d = i(\frac{K-1}{K})(\frac{N}{K})d$ is used as follows: every data point $D \in \mathcal{A}$ is divided across the dimension d into $\binom{K}{i}$ non-overlapping parts of size $d/\binom{K}{i}$ bits each, and then each part is labeled by a unique set $\mathcal{W} \subseteq [1 : K]$ of size i. The worker w_k stores the sub-point $D_{\mathcal{W}}$ in the excess storage, where $D \notin \mathcal{A}^t(k)$, only if $k \in \mathcal{W}$. Therefore, the number of sub-points a worker w_k is storing from a point $D \notin \mathcal{A}^t(k)$; is given by $\binom{K-1}{i-1}$ sub-points. The total number of these points is $N - \frac{N}{K} = \frac{(K-1)N}{K}$ points. Then, the total size necessary for excess storage is

$$\frac{(K-1)N}{K} \times \binom{K-1}{i-1} \times \frac{d}{\binom{K}{i}} = \frac{i(K-1)N}{K^2}d = \left(S - \frac{N}{K}\right)d, \tag{59}$$

which satisfies the memory constraint.

B. Data Delivery Phase

Next, we present our proposed delivery scheme to satisfy the new data assignment characterized by the shuffles (π_t, π_{t+1}). According to the adopted placement strategy, whenever a new data point is newly assigned to a worker, it already has $\binom{K-1}{i-1}$ out of the total $\binom{K}{i}$ partitions. Therefore, the number of sub-points still needed for every new assigned data point is $\binom{K}{i} - \binom{K-1}{i-1} = \binom{K-1}{i}$. This gives the total number of data sub-points needed by each worker to be $\binom{K-1}{i} \times |A^{t+1}(k) \setminus A^t(k)|$, where $|A^{t+1}(k) \setminus A^t(k)|$ is the number of data-points newly assigned to worker w_k at time $t+1$.

According to the placement strategy, every data sub-point $D_{\mathcal{W}}$, is stored at least in i different workers. Now, if we pick any set $\mathcal{M} \subseteq [1 : K]$ of the workers, where $|\mathcal{M}| = i + 1$, then for each worker w_k, where $k \in \mathcal{M}$, and for each point D newly assigned to w_k in the next shuffle, i.e., $D \notin \mathcal{A}^t(k)$, and $D \in \mathcal{A}^{t+1}(k)$, there is at least one sub-point needed by w_k from the remaining workers in the set, labeled as $D_{\mathcal{M}\setminus k}$. Therefore, we can send order $i+1$ coded symbols in the form $\bigoplus_{k\in\mathcal{M}} \left(A^{t+1}_{\mathcal{M}\setminus k}(k) \setminus A^t(k)\right)$, of size $d/\binom{K}{i}$ each, useful for all the $i+1$ workers in $\mathcal{M}$ in the same time. Note that in general, the data batches $A^{t+1}(k) \setminus A^t(k)$ differ in their sizes for different k, so we zero-pad the shorter batches before summing. Therefore, the number of such order $i+1$ coded symbols is $\max_{k\in\mathcal{M}} \quad |A^{t+1}(k)\setminus A^t(k)|$, of size $d/\binom{K}{i}$ each.

Considering all the possible sets $\mathcal{M}$ of size $i+1$, this process is repeated $\binom{K}{i+1}$ number of times, which gives the following coded symbols:

$$X_{\pi_t,\pi_{t+1}} = \left\{ \bigcup_{\substack{\mathcal{M}\subseteq[1:K]\\|\mathcal{M}|=i+1}} \bigoplus_{k\in\mathcal{M}} \left(A^{t+1}_{\mathcal{M}\setminus k}(k) \setminus A^t(k)\right) \right\}. \tag{60}$$

The corresponding total number of bits sent over the shared link is given by

$$R_{(\pi_t,\pi_{t+1})}d = \sum_{\substack{\mathcal{M}\subseteq[1:K]\\|\mathcal{M}|=i+1}} \max_{k\in\mathcal{M}} \quad |A^{t+1}(k)\setminus A^t(k)| \times \frac{d}{\binom{K}{i}}. \tag{61}$$

It is important to notice that the total number of times w_k becomes a member of the set $\mathcal{M}$ in (60) is $\binom{K-1}{i}$. Therefore, every worker gets $\binom{K-1}{i}\times|A^{t+1}(k)\setminus A^t(k)|$ sub-points in total, which are enough to recover the new assigned data points in $A^{t+1}(k)\setminus A^t(k)$ as previously discussed.

According to (60), the proposed coded shuffling scheme first finds the number of newly assigned data points to each worker w_k, i.e., $|A^{t+1}(k)\setminus A^t(k)|$. The probability that a data point D is newly assigned to the worker w_k is given as $\Pr(D\in A^{t+1}(k)\setminus A^t(k)) = \frac{K-1}{K}$. Therefore, as the number of points N grows larger we have

$$|A^{t+1}(k)\setminus A^t(k)| = \frac{K-1}{K}\frac{N}{K} + o(N). \tag{62}$$

Therefore using (61), we obtain the following rate for large values of N:

$$\begin{aligned}&\lim_{N\to\infty} R_{(\pi_t,\pi_{t+1})}d\\ &= \sum_{\substack{\mathcal{M}\subseteq[1:K]\\|\mathcal{M}|=i+1}} \frac{K-1}{K}\frac{N}{K}\times\frac{d}{\binom{K}{i}} = \frac{K-1}{K}\frac{N(K-i)}{K(i+1)}d.\end{aligned} \tag{63}$$

Worst-case Analysis of the Proposed Scheme: For the worst-case scenario, each worker is assigned completely new data points, and there are $\frac{N}{K}$ new data points for each worker, i.e., $\mathcal{A}^{t+1}(k)\setminus\mathcal{A}^t(k) = \mathcal{A}^{t+1}(k)$ and the number of new assigned data points is $|A^{t+1}(k)\setminus A^t(k)| = \frac{N}{K}$. Therefore using (60), the worst-case transmission is given as follows:

$$X_{\text{worst-case}} = \left\{ \bigcup_{\substack{\mathcal{M}\subseteq[1:K]\\|\mathcal{M}|=i+1}} \bigoplus_{k\in\mathcal{M}} A^{t+1}_{\mathcal{M}\setminus k}(k) \right\}. \tag{64}$$

Using (61), the corresponding total worst-case number of bits sent over the shared link is given by

$$\begin{aligned}R_{\text{worst-case}}d &= \sum_{\substack{\mathcal{M}\subseteq[1:K]\\|\mathcal{M}|=i+1}} \frac{N}{K}\times\frac{d}{\binom{K}{i}}\\ &= \binom{K}{i+1}\times\frac{N}{K}\times\frac{d}{\binom{K}{i}} = \frac{N(K-i)}{K(i+1)}d.\end{aligned} \tag{65}$$

Using the memory sharing concept in Claim 1, we can achieve the lower convex envelope of the following $K+1$ points for all $i\in[0:K]$:

$$\left(S=\left(1+i\frac{K-1}{K}\right)\frac{N}{K},\ R^{\text{upper}}_{\text{worst-case}} = \frac{N(K-i)}{K(i+1)}\right), \tag{66}$$

which completes the proof of Theorem 1.

Comparing (63) and (65), we notice that ratio between the rate for any arbitrary shuffle as the number of data points grows large and the worst-case rate is given as $\frac{K-1}{K}$.

C. Storage Update Procedure

In order to maintain the structure of the storage after the next shuffle at time $t+1$, the storage update procedure takes place at worker w_k for every point $D\in\mathcal{A}$ according to the following cases:

- <u>$D\in\mathcal{A}^t(k)$, and $D\in\mathcal{A}^{t+1}(k)$:</u> In this case D remains stored completely in Z_k^{t+1}.
- <u>$D\notin\mathcal{A}_k^t$, and $D\in\mathcal{A}_k^{t+1}$</u>: After the data delivery, worker w_k stores D completely in Z_k^{t+1}.
- <u>$D\in\mathcal{A}^t(k)$, and $D\notin\mathcal{A}^{t+1}(k)$:</u> Out of the point D previously stored completely in Z_k^t, worker w_k chooses to stores in Z_k^{t+1} the sub-points $D_{\mathcal{W}}$ where $k\in\mathcal{W}$.
- <u>$D\notin\mathcal{A}^t(k)$, and $D\notin\mathcal{A}^{t+1}(k)$:</u> Nothing changes about the storage of D in the excess storage of Z_k^{t+1}, and w_k keeps the same sub-points of D previously stored in Z_k^t, i.e., $D_{\mathcal{W}}$ where $k\in\mathcal{W}$.

Appendix II
Lower Bound on $R^*_{\text{worst-case}}$ (Proof of Theorem 2)

In this section, we present an information theoretic lower bound on the worst-case communication rate. We start by considering the following shuffle (π_t,π_{t+1}) at time $t+1$: for a permutation of the worker indexes $\sigma:(1,2,\ldots,K)\to(\sigma_1,\sigma_2,\ldots,\sigma_K)$, the worker w_{σ_k} at time $t+1$ is assigned the data batch that was assigned to the worker $w_{\sigma_{k-1}}$ at time t, i.e., $\mathcal{A}^{t+1}(\sigma_k) = \mathcal{A}^t(\sigma_{k-1})$, which gives the following condition using the decodability constraint in (10):

$$\begin{aligned}&H(\mathcal{A}^{t+1}(\sigma_k)|Z^t_{\sigma_k},X_{\pi_t,\pi_{t+1}})\\ &\quad= H(\mathcal{A}^t(\sigma_{k-1})|Z^t_{\sigma_k},X_{\pi_t,\pi_{t+1}}) = 0,\ \forall k\in[1:K].\end{aligned} \tag{67}$$

Furthermore using the uncoded storage contents at time t given in (13), for the worker w_{σ_k} the storage content is given as follows:

$$Z^{t+1}_{\sigma_k} = \left\{\mathcal{A}^t(\sigma_k),\ \bigcup_{j\in[1:K]\setminus k}\mathcal{A}^t(\sigma_j,\sigma_k)\right\}, \tag{68}$$

which also gives the following constraint:

$$H(\mathcal{A}^t(\sigma_k)|Z^t_{\sigma_k}) = 0,\quad \forall k\in[1:K]. \tag{69}$$

Note that the conditions (67) and (69) fully characterize the shuffle (π_t,π_{t+1}). Next, we prove that $H(\mathcal{A}|\mathbf{Z}^t_{\sigma_{[2:K]}},X_{\pi_t,\pi_{t+1}}) = 0$ using (69), and (67) as follows:

$$\begin{aligned}&H(\mathcal{A}|\mathbf{Z}^t_{\sigma_{[2:K]}},X_{\pi_t,\pi_{t+1}})\\ &= H(\mathcal{A}^t([1:K])|\mathbf{Z}^t_{\sigma_{[2:K]}},X_{\pi_t,\pi_{t+1}})\end{aligned}$$

$$\leq \sum_{j=1}^{K} H(\mathcal{A}^t(\sigma_j)|\mathbf{Z}^t_{\sigma_{[2:K]}}, X_{\pi_t,\pi_{t+1}})$$
$$\leq \sum_{j=2}^{K} H(\mathcal{A}^t(\sigma_j)|Z^t_{\sigma_j}) + H(\mathcal{A}^t(\sigma_1)|Z^t_{\sigma_2}, X_{\pi_t,\pi_{t+1}}) = 0. \tag{70}$$

Using (67), (69) and (70), we obtain the following bound:

$$\begin{aligned}
Nd &= H(\mathcal{A}) \\
&= I\left(\mathcal{A}; \mathbf{Z}^t_{\sigma_{[2:K]}}, X_{\pi_t,\pi_{t+1}}\right) + H\left(\mathcal{A}|\mathbf{Z}^t_{\sigma_{[2:K]}}, X_{\pi_t,\pi_{t+1}}\right) \\
&\overset{(a)}{=} I\left(\mathcal{A}; \mathbf{Z}^t_{\sigma_{[2:K]}}, X_{\pi_t,\pi_{t+1}}\right) \overset{(b)}{=} H\left(\mathbf{Z}^t_{\sigma_{[2:K]}}, X_{\pi_t,\pi_{t+1}}\right) \\
&\overset{(c)}{=} H\left(X_{\pi_t,\pi_{t+1}}, Z^t_{\sigma_K}\right) + \sum_{i=2}^{K-1} H\left(Z^t_{\sigma_i}|\mathbf{Z}^t_{\sigma_{[i+1:K]}}, X_{\pi_t,\pi_{t+1}}\right) \\
&\overset{(d)}{\leq} H\left(X_{\pi_t,\pi_{t+1}}\right) + H\left(Z^t_{\sigma_K}\right) \\
&\quad + \sum_{i=2}^{K-1} H\left(Z^t_{\sigma_i}|\mathbf{Z}^t_{\sigma_{[i+1:K]}}, X_{\pi_t,\pi_{t+1}}, \mathcal{A}^t(\sigma_{[i:K]})\right),
\end{aligned} \tag{71}$$

where (a) follows from (70), (b) follows from (4), and (8), where $\mathbf{Z}^t_{\sigma_{[2:K]}}$, and $X_{\pi_t,\pi_{t+1}}$ are deterministic functions of the data-set $\mathcal{A}$, (c) follows from the chain rule of entropy, and (d) follows from the processing constraint in (69) where the storage variables $\mathbf{Z}^t_{\sigma_{[i+1:K]}}$ contain $\mathcal{A}^t(\sigma_{[i+1:K]}$, and the decodability constraint in (67) where $Z^t_{\sigma_{i+1}}$ and $X_{\pi_t,\pi_{t+1}}$ can decode $\mathcal{A}^t(\sigma_{i+1})$. Using the storage contents in (68), we can write the previous bound as follows

$$\begin{aligned}
Nd &\overset{(a)}{\leq} R^*_{\pi_t,\pi_{t+1}}d + H\left(\mathcal{A}^t(\sigma_K)\right) \\
&\quad + H\left(\mathcal{A}^t(\sigma_{[1:K-1]}, \sigma_K)\right) \\
&\quad + \sum_{i=2}^{K-1} H\left(\mathcal{A}^t(\sigma_i), \mathcal{A}^t(\sigma_{[1:K]\setminus i}, \sigma_i)|\right. \\
&\qquad\qquad \left.\mathbf{Z}^t_{\sigma_{[i+1:K]}}, X_{\pi_t,\pi_{t+1}}, \mathcal{A}^t(\sigma_{[i:K]})\right) \\
&\overset{(b)}{=} R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d + H\left(\mathcal{A}^t(\sigma_{[1:K-1]}, \sigma_K)\right) \\
&\quad + \sum_{i=2}^{K-1} H\left(\mathcal{A}^t(\sigma_{[1:i-1]}, \sigma_i)|\right. \\
&\qquad\qquad \left.\mathbf{Z}^t_{\sigma_{[i+1:K]}}, X_{\pi_t,\pi_{t+1}}, \mathcal{A}^t(\sigma_{[i:K]})\right) \\
&\overset{(c)}{\leq} R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d + H\left(\mathcal{A}^t(\sigma_{[1:K-1]}, \sigma_K)\right) \\
&\quad + \sum_{i=2}^{K-1} H\left(\mathcal{A}^t(\sigma_{[1:i-1]}, \sigma_i)|\mathbf{Z}^t_{\sigma_{[i+1:K]}}\right) \\
&= R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d \\
&\quad + \sum_{i=2}^{K}\sum_{j=1}^{i-1} H\left(\mathcal{A}^t(\sigma_j, \sigma_i)|\mathbf{Z}^t_{[\sigma_{i+1}:\sigma_K]}\right) \\
&\overset{(d)}{=} R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d \\
&\quad + \sum_{i=2}^{K}\sum_{j=1}^{i-1} H\left(\mathcal{A}^t(\sigma_j, \sigma_i)|\mathcal{A}^t(\sigma_j, \sigma_{[i+1:K]})\right) \\
&= R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d \\
&\quad + \sum_{j=1}^{K-1}\sum_{i=j+1}^{K} H\left(\mathcal{A}^t(\sigma_j, \sigma_i)|\mathcal{A}^t(\sigma_j, \sigma_{[i+1:K]})\right) \\
&\overset{(e)}{=} R^*_{\pi_t,\pi_{t+1}}d + \frac{N}{K}d + \sum_{j=1}^{K-1} H\left(\mathcal{A}^t(\sigma_j, \sigma_{[j+1:K]})\right),
\end{aligned} \tag{72}$$

where (a) follows from the uncoded storage contents given in (68) where every worker w_k stores the assigned data batch $\mathcal{A}^t(k)$ plus separate functions of all the remaining data batches, i.e. $\mathcal{A}^t(j,k)$ for $j \in [1:K] \setminus k$, (b) is because after knowing $\mathcal{A}^t(\sigma_{[i:K]})$, the only parts left in $Z^t_{\sigma_i}$ are $\mathcal{A}^t(\sigma_{[1:i-1]}, \sigma_i)$, (c) because conditioning reduces entropy, (d) follows since $\mathcal{A}^t(\sigma_j, \sigma_i)$ only depends on the parts of the batch $\mathcal{A}^t(\sigma_j)$ stored at $\mathbf{Z}^t_{[\sigma_{i+1}:\sigma_K]}$, and finally (e) follows from the chain rule of entropy. From the definition in (14), we can write $\mathcal{A}^t(\sigma_j, \sigma_{[j+1:K]})$ as

$$\mathcal{A}^t(\sigma_j, \sigma_{[j+1:K]}) = \bigcup_{\mathcal{S}\subseteq\sigma_{[j+1:K]}:\, \mathcal{S}\neq\phi}\ \bigcup_{\mathcal{W}\subseteq[1:K]\setminus\sigma_j:\, \mathcal{S}\in\mathcal{W}} \mathcal{A}^t_{\mathcal{W}}(\sigma_j) \tag{73}$$

Therefore, we can upper bound the entropy $H\left(\mathcal{A}^t(\sigma_j, \sigma_{[j+1:K]})\right)$ as

$$\begin{aligned}
&H\left(\mathcal{A}^t(\sigma_j, \sigma_{[j+1:K]})\right) \\
&\quad\leq \sum_{\mathcal{S}\subseteq\sigma_{[j+1:K]}:\, \mathcal{S}\neq\phi}\ \sum_{\mathcal{W}\subseteq[1:K]\setminus\sigma_j:\, \mathcal{S}\in\mathcal{W}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)|\, d \\
&\quad= \sum_{\mathcal{W}\subseteq[1:K]\setminus\sigma_j} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)|\, d \ - \sum_{\mathcal{W}\subseteq\sigma_{[1:j-1]}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)|\, d,
\end{aligned} \tag{74}$$

where $|\mathcal{A}^t_{\mathcal{W}}(j)|$ is the size of the sub-batch $\mathcal{A}^t_{\mathcal{W}}(j)$ normalized by d. Therefore, by applying (74) in (72), we get a lower bound on $R^*_{\pi_t,\pi_{t+1}}$, which is also a lower bound on $R^*_{\text{worst-case}}$ following Remark 1, as follows:

$$\begin{aligned}
&R^*_{\text{worst-case}} \geq R_{\pi_t,\pi_{t+1}} \\
&\quad\geq N - \frac{N}{K} - \sum_{j=1}^{K-1}\left[\sum_{\mathcal{W}\subseteq[1:K]\setminus\sigma_j} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| \right. \\
&\qquad\qquad \left. - \sum_{\mathcal{W}\subseteq\sigma_{[1:j-1]}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)|\right] \\
&\quad= N - \frac{N}{K} - \sum_{\ell=0}^{K-1}\sum_{j=1}^{K-1}\left[\sum_{\substack{\mathcal{W}\subseteq[1:K]\setminus\sigma_j \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| \right. \\
&\qquad\qquad \left. - \sum_{\substack{\mathcal{W}\subseteq\sigma_{[1:j-1]} \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)|\right].
\end{aligned} \tag{75}$$

For $K!$ possible permutations σ of the ordered set $(1, 2, \ldots, K)$, we get $K!$ different bounds over $R^*_{\text{worst-case}}$ from (75). Summing up over all the possible $K!$ permutations σ, we get

$$R^*_{\text{worst-case}} \geq N - \frac{N}{K} - \frac{1}{K!} \sum_{\ell=0}^{K-1} \sum_{j=1}^{K-1} \sum_{\sigma \in [K!]} \left[\sum_{\substack{\mathcal{W} \subseteq [1:K] \setminus \sigma_j \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| - \sum_{\substack{\mathcal{W} \subseteq \sigma_{[1:j-1]} \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| \right], \tag{76}$$

where $[K!]$ is defined as the set of all possible permutations of the ordered set $(1, 2, \ldots, K)$, which contains $K!$ permutations. Due to symmetry, for each value of a (ℓ, j) pair in the outer summation in (76), where $\ell \in [0 : K-1]$ and $j \in [1 : K-1]$, the coefficients of each $|\mathcal{A}^t_{\mathcal{W}}(k)|$ in the inner summation for $k \in [1 : K]$ and $|\mathcal{W}| = \ell$ are equal. Therefore, we can write the inner summation in (76) in the following form:

$$\begin{aligned} &\sum_{\sigma \in [K!]} \left[\sum_{\substack{\mathcal{W} \subseteq [1:K] \setminus \sigma_j \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| - \sum_{\substack{\mathcal{W} \subseteq \sigma_{[1:j-1]} \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| \right] \\ &= \left(c_1^{j,\ell} - c_2^{j,\ell} \right) \sum_{k=1}^{K} \sum_{\mathcal{W} \subseteq [1:K]:\, |\mathcal{W}|=\ell} |\mathcal{A}^t_{\mathcal{W}}(k)| \\ &= \left(c_1^{j,\ell} - c_2^{j,\ell} \right) x_\ell, \end{aligned} \tag{77}$$

where $c_1^{j,\ell}$, and $c_2^{j,\ell}$ are the two coefficients of x_ℓ coming from the two inner summations in the LHS of (77). From (77), finding $c_1^{j,\ell}$, and $c_2^{j,\ell}$ is the same as finding the coefficients of one realization of k, and $\mathcal{W}$ on the right side of the equation, and we consider for instance $\mathcal{A}^t_{[2:\ell+1]}(1)$. In the first sum, we get $c_1^{j,\ell}$ by counting the number of permutations where $\sigma_j = 1$, which is given by

$$c_1^{j,\ell} = (K-1)!. \tag{78}$$

In the second sum, we get $c_2^{j,\ell}$ by counting the number of permutations such that $\sigma_j = 1$, and $\sigma_{j+1}, \ldots, \sigma_K \in [\ell + 2 : K]$, which is given by

$$c_2^{j,\ell} = \frac{(K-\ell-1)!}{(j-\ell-1)!}(j-1)! = \frac{\binom{j-1}{\ell}}{\binom{K-1}{\ell}}(K-1)!. \tag{79}$$

Therefore, we can write the summation in (77) in the following form:

$$\begin{aligned} &\sum_{\sigma \in [K!]} \left[\sum_{\substack{\mathcal{W} \subseteq [1:K] \setminus \sigma_j \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| - \sum_{\substack{\mathcal{W} \subseteq \sigma_{[1:j-1]} \\ |\mathcal{W}|=\ell}} |\mathcal{A}^t_{\mathcal{W}}(\sigma_j)| \right] \\ &= \left((K-1)! - \frac{\binom{j-1}{\ell}}{\binom{K-1}{\ell}}(K-1)! \right) x_\ell. \end{aligned} \tag{80}$$

Now, we use (80) in (76) to obtain the following bound:

$$\begin{aligned} R^*_{\text{worst-case}} &\geq N - \frac{N}{K} - \frac{1}{K!} \sum_{\ell=0}^{K-1} \sum_{j=1}^{K-1} \left[(K-1)! - \frac{\binom{j-1}{\ell}}{\binom{K-1}{\ell}}(K-1)! \right] x_\ell \\ &= N - \frac{N}{K} - \frac{1}{K!} \sum_{\ell=0}^{K-1} \left[(K-1)(K-1)! - \frac{\binom{K-1}{\ell+1}}{\binom{K-1}{\ell}}(K-1)! \right] x_\ell \\ &= N - \frac{N}{K} - \frac{1}{K} \sum_{\ell=0}^{K-1} \left[(K-1) - \frac{K-\ell-1}{t+1} \right] x_\ell \\ &\overset{(a)}{=} \sum_{\ell=0}^{K-1} x_\ell - \frac{N}{K} - \sum_{\ell=0}^{K-1} \frac{\ell}{\ell+1} x_\ell = \sum_{\ell=0}^{K-1} \frac{1}{\ell+1} x_\ell - \frac{N}{K}, \end{aligned} \tag{81}$$

where (a) follows from the data size constraint in (15). Next, we obtain $K-1$ different lower bounds on the optimal worst-case transmission rate R^*_{wc}, by eliminating the pairs (x_{j-1}, x_j), for each $j \in [1 : K-1]$, in the equation (81) using the equations (15) and (17). We use (15) to write x_{j-1} as follows:

$$x_{j-1} = N - \sum_{\ell \in [0:K] \setminus j-1} x_\ell. \tag{82}$$

We first apply (82) in (81) to obtain

$$\begin{aligned} R^*_{\text{worst-case}} &\geq \sum_{\ell \in [0:K-1] \setminus j-1} \frac{1}{\ell+1} x_\ell + \frac{1}{j} \left(N - \sum_{\ell \in [0:K-1] \setminus j-1} x_\ell \right) - \frac{N}{K} \\ &= \frac{N(K-j)}{Kj} - \sum_{\ell \in [0:K-1] \setminus j-1} \frac{\ell - j + 1}{j(\ell+1)} x_\ell. \end{aligned} \tag{83}$$

We next apply (82) in the excess storage constraint of (17) to obtain

$$\begin{aligned} &\sum_{\ell \in [0:K-1] \setminus j-1} \ell x_\ell + (j-1) \left(N - \sum_{\ell \in [0:K] \setminus j-1} x_\ell \right) \leq K \left(S - \frac{N}{K} \right), \\ &\sum_{\ell \in [0:K-1] \setminus j-1} (\ell - j + 1) x_\ell \leq K \left(S - j\frac{N}{K} \right). \end{aligned} \tag{84}$$

Now, we need to eliminate x_j from (83). We use (84) to bound x_j as

$$x_j \leq K \left(S - j\frac{N}{K} \right) - \sum_{\ell \in [0:K-1] \setminus \{j-1, j\}} (\ell - j + 1) x_\ell. \tag{85}$$

Then, we use this bound in (83) as follows:

$$R^*_{\text{worst-case}} \geq \frac{N(K-j)}{Kj} - \sum_{\ell \in [0:K-1] \setminus \{j-1, j\}} \frac{\ell - j + 1}{j(\ell+1)} x_\ell - \frac{1}{j(j+1)} x_j$$

$$\overset{(a)}{\geq} \frac{N(K-j)}{Kj} - \sum_{\ell\in[0:K-1]\setminus\{j-1,j\}} \frac{\ell-j+1}{j(\ell+1)}x_\ell - \frac{K\left(S-j\frac{N}{K}\right)}{j(j+1)} + \sum_{\ell\in[0:K-1]\setminus\{j-1,j\}} \frac{(\ell-j+1)}{j(j+1)}x_\ell$$
$$= \frac{N(K-j)}{Kj} - \frac{K\left(S-j\frac{N}{K}\right)}{j(j+1)} + \sum_{\ell\in[0:K-1]\setminus\{j-1,j\}} \lambda_\ell x_\ell$$
$$\overset{(b)}{\geq} \frac{N(K-j)}{Kj} - \frac{K\left(S-j\frac{N}{K}\right)}{j(j+1)}, \tag{86}$$

where (a) follows from (85) where the coefficient of x_j in the above equation is negative for all $j \in [1:K-1]$, and (b) since the coefficients, λ_ℓ, of $x_\ell > 0$ are positive for $\ell \in [0:K-1]\setminus\{j-1,j\}$, which can be shown in the following:

$$\lambda_\ell = \frac{\ell-j+1}{j(j+1)} - \frac{\ell-j+1}{j(\ell+1)} = \frac{(\ell-j)(\ell-j+1)}{j(j+1)(\ell+1)}, \tag{87}$$

where $j, j+1, \ell+1 > 0$ for $\ell, j \geq 0$, then we only need to show that $(\ell-j)(\ell-j+1) > 0$ for $\ell \in [0:K-1]\setminus\{j-1,j\}$. This can be easily checked by assuming $y = \ell - j$, then $y(y+1)$ is only negative in the range $-1 < y < 0$, or $j-1 < \ell < j$, which is not in the range of ℓ in the above summation.

The lower bound in (86) is a linear function of S for a fixed value of $j \in [1:K-1]$ passing through the points $\left(S_1 = j\frac{N}{K}, R_1 = \frac{N(K-j)}{Kj}\right)$, and $\left(S_2 = (j+1)\frac{N}{K}, R_2 = \frac{N(K-j-1)}{K(j+1)}\right)$. We obtain $K-1$ such lower bounds for every $j \in [1:K-1]$, which eventually give the lower bound on $R^*_{\text{worst-case}}$ as the lower convex envelope of the following K points:

$$\left(S = m\frac{N}{K},\ R^{\text{lower}}_{\text{worst-case}} = \frac{N(K-m)}{Km}\right), \quad \forall m \in [1:K], \tag{88}$$

which completes the proof of Theorem 2.

Appendix III
Maximum Gap Analysis (Proof of Theorem 3)

To characterize the maximum gap between the obtained bounds over $R^*_{\text{worst-case}}$, we first express the storage S as multiples of $\frac{N}{K}$, i.e., $S = m\frac{N}{K}$, for $1 \leq m \leq K$. From Theorem 1 for $\left(1+i\frac{K-1}{K}\right) \leq m \leq \left(1+(i+1)\frac{K-1}{K}\right)$, and $i \in [0:K-1]$, we can achieve the line joining the two points $\left(m = \left(1+i\frac{K-1}{K}\right), R = \frac{N(K-i)}{K(i+1)}\right)$, and $\left(m = \left(1+(i+1)\frac{K-1}{K}\right), R = \frac{N(K-i-1)}{K(i+2)}\right)$, which gives the following upper bounds over $R^*_{\text{worst-case}}$ as

$$\frac{R^{\text{upper}}_{\text{worst-case}} - \frac{N(K-i)}{K(i+1)}}{m - \left(1+i\frac{K-1}{K}\right)} = \frac{\frac{N(K-i-1)}{K(i+2)} - \frac{N(K-i)}{K(i+1)}}{\left(1+(i+1)\frac{K-1}{K}\right) - \left(1+i\frac{K-1}{K}\right)} = -\frac{N(K+1)}{(K-1)(i+1)(i+2)},$$
$$R^{\text{upper}}_{\text{worst-case}} = \frac{N(K-i)}{K(i+1)} - \frac{N(K+1)}{(K-1)(i+1)(i+2)}\left(m-1-i\frac{K-1}{K}\right), \tag{89}$$

for $\left(1+i\frac{K-1}{K}\right) \leq m \leq \left(1+(i+1)\frac{K-1}{K}\right)$, and $i \in [0:K-1]$. Also, from (86) we have the lower bounds over $R^*_{\text{worst-case}}$ as

$$R^{\text{lower}}_{\text{worst-case}} = \frac{N(K-j)}{Kj} - \frac{N(m-j)}{j(j+1)}, \tag{90}$$

for $j \leq m \leq j+1$, and $j \in [1:K-1]$.

Due to the properties of the piece-wise linear functions, we obtain the maximum gap at one of the following $2K-1$ values of m: $m = j$, for $j \in [1:K-1]$, or $m = 1+i\frac{K-1}{K}$, for $i \in [1:K]$.

A. Gap Analysis for $m = 1+i\frac{K-1}{K}$, and $i \in [1:K]$

We first notice that when $i \in [1:K]$, then $i \leq m \leq i+1$. Therefore, the lower bound $R^{\text{lower}}_{\text{worst-case}}$ at $m = 1+i\frac{K-1}{K}$ follows from (90) where $j = i$:

$$R^{\text{lower}}_{\text{worst-case}}\left(m = 1+i\frac{K-1}{K}\right) = \frac{N(K-i)}{Ki} - \frac{N\left(1+i\frac{K-1}{K}-i\right)}{i(i+1)} = \frac{N(K-i)}{Ki} - \frac{N(K-i)}{Ki(i+1)} = \frac{N(K-i)}{K(i+1)}, \tag{91}$$

which matches the upper bound in (89), when $m = 1+i\frac{K-1}{K}$. Therefore, the proposed achievable scheme is optimal for $m = 1+i\frac{K-1}{K}$, where $i \in [1:K]$.

B. Gap Analysis for $m = j$, and $j \in [1:K-1]$

We first notice that when $m = j$, then $\left(1+(j-1)\frac{K-1}{K}\right) \leq m \leq \left(1+j\frac{K-1}{K}\right)$ for $j \in [1:K-1]$. Therefore, the upper bound $R^{\text{upper}}_{\text{worst-case}}$ at $m = j$ follows from (89) where $i = j-1$:

$$R^{\text{upper}}_{\text{worst-case}}(m=j) = \frac{N(K-j+1)}{Kj} - \frac{N(K+1)}{j(K-1)(j+1)}\left(j-1-(j-1)\frac{K-1}{K}\right) = \frac{N(K-j+1)}{Kj} - \frac{N(K+1)(j-1)}{jK(K-1)(j+1)} = \frac{N(K-j)}{Kj} + \frac{N}{Kj}\left(1-\frac{(K+1)(j-1)}{(K-1)(j+1)}\right), \tag{92}$$

whereas the lower bound on $R^*_{\text{worst-case}}(m = j)$ follows from (90) directly as follows:

$$R^{\text{lower}}_{\text{worst-case}}(m=j) = \frac{N(K-j)}{Kj}. \tag{93}$$

Hence, the ratio between the bounds follows by dividing (92) by (93) as

$$\frac{R^{\text{upper}}_{\text{worst-case}}}{R^{\text{lower}}_{\text{worst-case}}} = 1 + \frac{1}{K-j}\left(1-\frac{(K+1)(j-1)}{(K-1)(j+1)}\right) = 1 + \frac{2}{(K-1)(j+1)}, \quad j \in [1:K-1]. \tag{94}$$

We notice that ratio in (94) is a decreasing function in j. Therefore, we obtain the maximum gap with the smallest value of j, i.e., $j = 1$, which is the no excess storage case $S = \frac{N}{K}$.

Applying $j = 1$ in (94), we obtain the maximum gap ratio as follows:

$$\frac{R^{\text{upper}}_{\text{worst-case}}}{R^{\text{lower}}_{\text{worst-case}}} = 1 + \frac{1}{(K-1)} = \frac{K}{K-1}, \tag{95}$$

which completes the proof of Theorem 3.

Appendix IV
Closing the Gap (Proof of Theorem 4)

Based on Example III-D.1, we introduce the general achievability to close the gap for some storage values. In particular, we consider the storage values $S = m\frac{N}{K}$, for $m \in \{1, K-2, K-1\}$, any number of workers K, and any number of data points N. We also consider a variation of the SIP mechanism which we refer to as the *modified SIP mechanism*.

A. Modified Structural Invariant Placement (Modified SIP)

Assume in the following that the dimensionality of the data points d is integer multiples of $\binom{K-1}{m-1}$. Every data point D_i for $i \in [1:N]$ is now partitioned into $\binom{K-1}{m-1}$ non-overlapped sub-points. As suggested in the Example III-D.1, the labeling for the data sub-points is changing over time as follows: At the time epoch t, the data sub-points of the data point D_i are labeled by unique subsets $\mathcal{W}_t \subseteq [1:K] \setminus \delta_t(i)$, where $\delta_t(i)$ is the index of the worker assigned to the data point D_i at time t. Every worker stores the assigned data points as well as the data sub-points having the worker's index in their labels. Therefore, any partition of a data point is stored at total number of m workers; $m-1$ workers are storing it as excess storage, and 1 worker is assigned the whole corresponding data point for processing.

For invariant structure placement, the change in the labels at time $t+1$ is required only for the data sub-points $D_{i,\mathcal{W}_t}$, where $\delta_{t+1}(i) \in \mathcal{W}_t$, by replacing $\delta_{t+1}(i)$ with $\delta_t(i)$ in the label $\mathcal{W}_t$ to obtain the newly labeled sub-points $D_{i,\mathcal{W}_{t+1}}$ where $\delta_t(i) \in \mathcal{W}_{t+1}$. Therefore, these newly labeled sub-points are required now to be stored in the excess storage of the worker $w_{\delta_t(i)}$, which already has the data point D_i fully available at its cache at time t. Then, there is no need to deliver these sub-points, and the storage structure can be preserved.

The number of data sub-points of the point D_i needed to be stored at worker w_k at time t, where $\delta_t(i) \neq k$ ($D_i \notin \mathcal{A}^t(k)$) is $\binom{K-2}{m-2}$ of size $d/\binom{K-1}{m-1}$ bits each. In total, we have $(K-1)\frac{N}{K}$ such data points where $\delta_t(i) \neq k$ for the worker w_k. Therefore, the worker w_k needs to store in the excess storage data of total size

$$\begin{aligned}(K-1)\frac{N}{K} \times \binom{K-2}{m-2} \times \frac{d}{\binom{K-1}{m-1}} \\ = (m-1)\frac{N}{K}d = \left(S - \frac{N}{K}\right)d,\end{aligned} \tag{96}$$

which satisfies the memory constraint.

Before we proceed to the delivery mechanism we define $\mathcal{A}^{t,t+1}(i;j) = \mathcal{A}^t(i) \cap \mathcal{A}^{t+1}(j)$ as the part of data assigned to w_j at time $t+1$ which was also assigned to w_i at time t. Furthermore, we define $S^{t,t+1}_{i,j} = |\mathcal{A}^{t,t+1}(i;j)|$ as the number of such data points. Therefore, the data batches $\mathcal{A}^t(i)$ and $\mathcal{A}^{t+1}(i)$ can then be written as

$$\begin{aligned}\mathcal{A}^t(i) = \cup_{j=1}^{K} \mathcal{A}^{t,t+1}(i;j), \\ \mathcal{A}^{t+1}(i) = \cup_{j=1}^{K} \mathcal{A}^{t,t+1}(j;i).\end{aligned} \tag{97}$$

Since we have the size of the data batches is fixed as $|\mathcal{A}^t(i)| = |\mathcal{A}^{t+1}(i)| = \frac{N}{K}$, we obtain the following property:

$$\sum_{j=1}^{K} S^{t,t+1}_{i,j} = \sum_{j=1}^{K} S^{t,t+1}_{j,i} = \frac{N}{K}. \tag{98}$$

Remark 2 (Data-flow Conservation Property): We next state an important property satisfied by any shuffle, namely the data-flow conservation property:

$$\sum_{j \in [1:K] \setminus i} S^{t,t+1}_{i,j} = \sum_{j \in [1:K] \setminus i} S^{t,t+1}_{j,i}. \tag{99}$$

The proof of this property follows directly by subtracting $S^{t,t+1}_{i,i}$ from the two sides of (98), and has the following interesting interpretation: the total number of new data points that need to be delivered to worker w_i (and are present elsewhere), i.e., the RHS of (99), is exactly equal to the total number of data points that worker w_i has that are desired by the other workers, which is the LHS of (99).

The rate $R_{\pi_t,\pi_{t+1}}$ is characterized by $S^{t,t+1}_{i,j}$ for $i, j \in [1:K]$. These shuffling parameters can be held in the matrix $S^{t,t+1} = [S^{t,t+1}_{i,j}]_{i,j}$, which can be named as the *shuffling matrix*. Moreover, according to the property in (98) the shuffling matrix $S^{t,t+1}$ is a $K \times K$ square matrix with the row sum equals the column sum equals $\frac{N}{K}$. In the following discussion, we drop the superscript $t, t+1$ from $\mathcal{A}^{t,t+1}(i;j)$, and $S^{t,t+1}_{i,j}$ for short notation.

Lemma 1: The rate achieved when the diagonal entries of the shuffling matrix are greater than zero, i.e., when $S_{i,i} > 0$ for $i \in [1:K]$, is no larger than the worst-case rate.

Proof: The proof is straight forward, where $S_{i,i}$ is the number of data points that are needed by worker w_i at times t and $t+1$. Therefore, they remain in the storage of the worker w_i and do not participate in the communication process. If $S_{i,i} > 0$, then less number of data points are needed by worker w_i and the rate is no larger than the worst-case rate, which completes the proof of the lemma. ■

Corollary 5: For the worst-case rate analysis, we can assume that every worker is assigned only new data points, i.e., $S_{i,i} = 0$. Hence, the data conservation property in (99) can be written as

$$\sum_{j \in [1:K] \setminus j} S_{i,j} = \sum_{j \in [1:K] \setminus j} S_{j,i} = \frac{N}{K}. \tag{100}$$

B. Closing the Gap for m = 1

We consider the storage value $m = 1$ ($S = \frac{N}{K}$), which is the no-excess storage case considered in our previous work [34] for any arbitrary shuffle. One can easily show that the pair ($S = \frac{N}{K}$, $R_{\text{worst-case}} = (K-1)\frac{N}{K}$) is achievable by sending $K-1$ linear independent combinations of the K data batches at time t, i.e., $\mathcal{A}^t(1), \ldots, \mathcal{A}^t(K)$, to satisfy any data assignment at time $t+1$. Since every worker w_k has already the data batch $\mathcal{A}^t(k)$ already stored in its cache, it can solve for the remaining $K-1$ batches and obtain the whole data-set to store the new data assignment.

C. Closing the Gap for $m = K - 1$

According to the adopted placement strategy, whenever a new data point is needed at any worker, it already has $\binom{K-2}{m-2}$ out of the total $\binom{K-1}{m-1}$ partitions, that is for the storage value $m = K-1$ ($S = (K-1)\frac{N}{K}$), only 1 out of $K-1$ sub-points is needed. Furthermore, this needed data sub-point is already available at the remaining $m = K-1$ workers. Therefore, for the $S_{i,j}$ data points assigned to worker w_j and available at w_i, i.e., $\mathcal{A}(i;j)$, the data sub-batch $\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j)$ is the only part needed to be transmitted to w_j, which is available at all the workers except w_j. For the worst-case scenario according to Corollary 5, we assume every worker is assigned completely new data batch, i.e., $S_{i,i} = 0$ for all $i \in [1:K]$. Therefore, we can write the total part needed to be transmitted to w_j as $\cup_{i\in[1:K]\setminus j}\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j)$, which consists of $\frac{N}{K}$ data sub-points each of size $\frac{d}{K-1}$ each, and the size of $\cup_{i\in[1:K]\setminus j}\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j)$ (normalized by d) is

$$|\cup_{i\in[1:K]\setminus j}\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j)| = \frac{N}{K(K-1)}. \tag{101}$$

In the delivery phase, we can send the following coded data batch:

$$\bigoplus_{j\in[1:K]} \cup_{i\in[1:K]\setminus j}\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j), \tag{102}$$

which is useful for the K workers in the same time as follows: w_k has $\bigoplus_{j\in[1:K]\setminus k}\cup_{i\in[1:K]\setminus j}\mathcal{A}_{[1:K]\setminus\{i,j\}}(i;j)$ which it can subtract to recover the needed part $\cup_{i\in[1:K]\setminus k}\mathcal{A}_{[1:K]\setminus\{i,k\}}(i;k)$. Moreover, the size of the coded transmission in (102) is the same as the size of the uncoded elements given in (101) as $\frac{N}{K(K-1)}$, which achieves the pair ($S = (K-1)\frac{N}{K}$, $R_{\text{worst-case}} = \frac{N}{K(K-1)}$).

D. Closing the Gap for $m = K - 2$

For the storage point $m = K-2$ ($S = (K-2)\frac{N}{K}$), whenever a data point is newly assigned to a worker, it already has $\binom{K-2}{K-4} = \frac{(K-2)(K-1)}{2}$ out of $\binom{K-1}{K-3} = \frac{(K-1)(K-2)}{2}$ parts, and hence only $K-2$ parts are needed of size $\frac{2d}{(K-1)(K-2)}$ bits each. We also assume the worst-case scenario, where according to Corollary 5 every worker is assigned completely new data batch, i.e., $S_{i,i} = 0$ and worker w_i needs $\frac{N}{K}$ new data points for all $i \in [1:K]$. Therefore, the total number of sub-points needed by every worker is $(K-2)\frac{N}{K}$.

• Consider the data sub-points which are considered interference to w_k (neither available nor needed). First, w_k does not need nor previously assigned the data points in the batches $\mathcal{A}(i;j)$ where $i \neq j$ and $i,j \in [1:K]\setminus k$ (potential interference). However, not the whole data points in $\mathcal{A}(i;j)$ are sent to w_j, since w_j has already some parts of them, which are given by $\mathcal{A}_{\mathcal{W}}(i;j)$, where $j \in \mathcal{W}$ and $|\mathcal{W}| = K-3$. Moreover, w_k also has some parts available in its cache of $\mathcal{A}(i;j)$ given by $\mathcal{A}_{\mathcal{W}}(i;j)$, where $k \in \mathcal{W}$ (do not cause interference). As a summary, the part of $\mathcal{A}(i;j)$, where $i \neq j$ and $i,j \in [1:K]\setminus k$, that is considered interference to w_k is given by $\mathcal{A}_{[1:K]\setminus\{i,j,k\}}(i;j)$, and hence the total interference faced by w_k is

$$\mathcal{I}(k) = \bigcup_{\substack{i,j\in[1:K]\setminus k\\ i\neq j}} \mathcal{A}_{[1:K]\setminus\{i,j,k\}}(i;j). \tag{103}$$

• Next, we organize these interference sub-batches according to the workers that need them as in Figure 10a. Worker w_j, where $j \in [1:K]\setminus k$, needs the following sub-batches causing interference to w_k:

$$\mathcal{I}(j;k) = \bigcup_{i\in[1:K]\setminus\{k,j\}} \mathcal{A}_{[1:K]\setminus\{i,j,k\}}(i;j), \tag{104}$$

which consists of data sub-points of size $\frac{2d}{(K-1)(K-2)}$ each and total number given by

$$\begin{aligned} I_{j;k} &= \sum_{i\in[1:K]\setminus\{k,j\}} S_{i,j} \\ &= N/K - S_{k,j} = \sum_{i\in[1:K]\setminus\{k,j\}} S_{k,i}. \end{aligned} \tag{105}$$

Note that $\mathcal{I}(j;k)$ serves as: a) interference to w_k, b) useful for w_j; and c) available at all the remaining workers. Also, the total interference faced by w_k can be written as $\mathcal{I}(k) = \cup_{j\in[1:K]\setminus k}\mathcal{I}(j;k)$ which consists of data sub-points of size $\frac{2d}{(K-1)(K-2)}$ bits each and total number given by

$$\begin{aligned} I_k = \sum_{j\in[1:K]\setminus k} I_{j;k} &= \sum_{j\in[1:K]\setminus k} (N/K - S_{k,j}) \\ &= (K-2)\frac{N}{K}. \end{aligned} \tag{106}$$

• Following Example III-D.1, we apply a similar interference alignment argument. We first break $\mathcal{I}(j;k)$ for every $j \in [1:K]\setminus k$ into $K-2$ partitions labeled as $\mathcal{I}^{(i)}(j;k)$ for $i \in [1:K]\setminus\{j,k\}$. The number of sub-points in $\mathcal{I}^{(i)}(j;k)$ is $S_{k,i}$ which satisfies the total size of $\mathcal{I}(j;k)$ given in (105). As shown in Figure 10b, we generate $S_{k,i}$ coded sub-points for every $i \in [1:K]\setminus k$ as follows:

$$\begin{aligned} &S_{k,i} \text{ coded sub-points}: \\ &\mathcal{C}^{(i)}(k) = \bigoplus_{j\in[1:K]\setminus\{k,i\}} \mathcal{I}^{(i)}(j;k), \quad \forall i \in [1:K]\setminus k. \end{aligned} \tag{107}$$

Note that $\mathcal{C}^{(i)}(k)$ is a coded sub-batch serves as: a) aligned interference to w_k, b) available at w_i as $j \neq i$ in the above summation; and c) useful for all the remaining workers as follows: worker w_ℓ for $\ell \notin \{i,k\}$ has $\bigoplus_{j\in[1:K]\setminus\{k,i,\ell\}}\mathcal{I}^{(i)}(j;k)$ so it can subtract from $\mathcal{C}^{(i)}(k)$ to get the needed part $\mathcal{I}^{(i)}(\ell;k)$.

• The total size of $\cup_{i\in[1:K]\setminus k}\mathcal{C}^{(i)}(k)$ is $\sum_{i\in[1:K]\setminus k} S_{k,i} = \frac{N}{K}$ coded sub-points, which aligns the $I_k = (K-2)\frac{N}{K}$ total interference sub-points seen by w_k, i.e., $\mathcal{I}(k)$ into $\frac{N}{K}$ coded sub-points. In the same time, these $\frac{N}{K}$ coded sub-points serve, for each remaining worker w_j for $j \neq k$, as $\sum_{i\in[1:K]\setminus\{j,k\}} S_{k,i} = \frac{N}{K} - S_{k,j}$ useful sub-points given by $\cup_{i\in[1:K]\setminus\{k,j\}}\mathcal{C}^{(i)}(k)$, while the remaining $S_{k,j}$ sub-points, given by $\mathcal{C}^{(j)}(k)$, are available at w_j's cache.

• By aligning all the interference seen by all the workers, i.e., generating the coded batches $\cup_{i\in[1:K]\setminus k}\mathcal{C}^{(i)}(k)$ for all

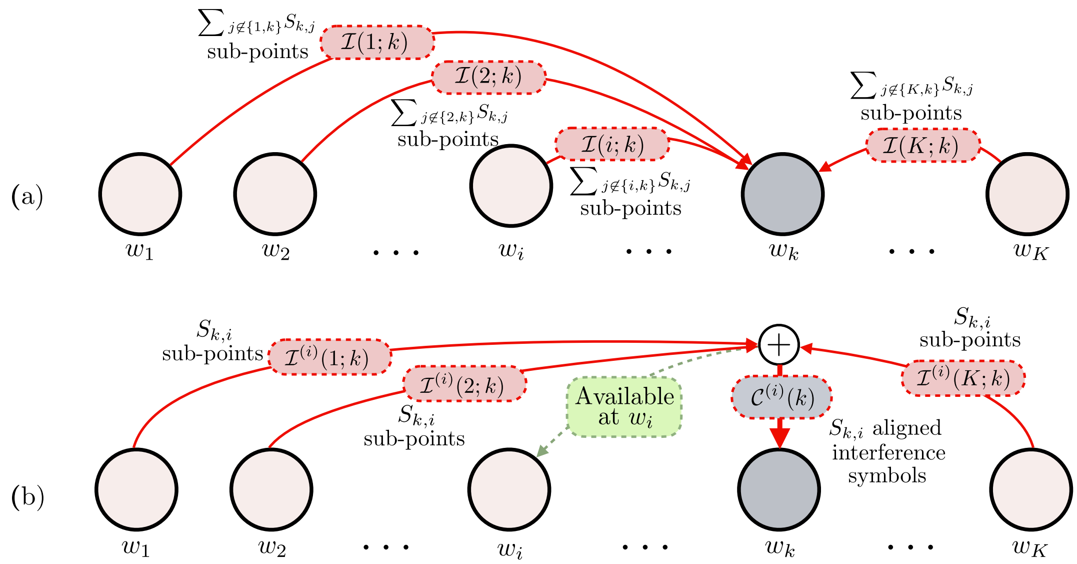

Fig. 10. In (a), the interference sub-batches seen by w_k are organized according to the workers that need them, such that $\mathcal{I}(i;k)$ for $i \neq k$ is the data needed to be delivered to w_i while causing interference to w_k. In (b), we pick $S_{k,i}$ sub-points from each $\mathcal{I}(j;k)$ (labeled as $\mathcal{I}^{(i)}(j;k)$) where $j \notin \{i,k\}$, and align them into $S_{k,i}$ coded symbols labeled by the set $\mathcal{C}^{(i)}(k)$, which serve as aligned interference for w_k, available at w_i, and useful for the remaining workers.

$k \in [1:K]$, we get a total number of N coded sub-points seen as follows by every worker w_j: a) $\frac{N}{K}$ aligned interference coded sub-points, b) $\sum_{k\in[1:K]\setminus j} S_{k,j} = \frac{N}{K}$ available sub-points; and c) $\sum_{k\in[1:K]\setminus j} \left(\frac{N}{K} - S_{k,j}\right) = (K-2)\frac{N}{K}$ useful sub-points, which satisfies the total number of sub-points needed in the worst case as discussed in the beginning. Since out of all the N coded sub-points every worker already has $\frac{N}{K}$ of them, then the N coded sub-points can be sent in only $(K-1)\frac{N}{K}$ linear independent combinations of size $\frac{2d}{(K-1)(K-2)}$ bits each, where the interference sub-points occupy $\frac{N}{K}$ dimensions, while the useful sub-points occupy $(K-2)\frac{N}{K}$ dimensions. As a result, the total rate achieved is $\frac{2N}{K(K-2)}d$ bits, which achieves the pair $(S = (K-2)\frac{N}{K}, R_{\text{worst-case}} = \frac{2N}{K(K-2)})$.

Now that we have closed the gap between the bounds in Theorems 1 and 2 for $S = m\frac{N}{K}$, where $m \in \{1, K-2, K-1\}$, which covers all the storage values for $K < 5$, while for $K \geq 5$ we can do the same analysis as in Section III to obtain the gap ratio similar to (94) as follows:

$$\frac{R^{\text{upper}}_{\text{worst-case}}}{R^{\text{lower}}_{\text{worst-case}}} = 1 + \frac{1}{K-j}\left(1 - \frac{(K+1)(j-1)}{(K-1)(j+1)}\right) = 1 + \frac{2}{(K-1)(j+1)}, \quad j \in [2:K-1], \quad (108)$$

which is maximized for $j = 2$ to obtain the maximum gap ratio as $1 + \frac{2}{(K-1)(3)} = \frac{K-\frac{1}{3}}{K-1}$ for $K \geq 5$ which completes the proof of Theorem 4.

References

[1] L. Bottou, "Stochastic gradient descent tricks," in *Proc. Neural Netw., Tricks Trade*, 2012, pp. 421–436.

[2] O. Shamir, "Without-Replacement Sampling for Stochastic Gradient Methods: Convergence Results and Application to Distributed Optimization," 2016, *arXiv:1603.00570*. [Online]. Available: https://arxiv.org/abs/1603.00570

[3] M. Gürbüzbalaban, A. Ozdaglar, and P. Parrilo, "Why random reshuffling beats stochastic gradient descent," 2015, *arXiv:1510.08560*. [Online]. Available: https://arxiv.org/abs/1510.08560

[4] J. Chung, K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Ubershuffle: Communication-efficient data shuffling for sgd via coding theory," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–7.

[5] S. Shi, G.-H. Hsu, K. Nahrstedt, and R. Campbell, "Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming," in *Proc. 19th ACM Int. Conf. Multimedia*, Dec. 2011, pp. 103–112.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[7] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[8] G. Ananthanarayanan *et al.*, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. USENIX OSDI*, Oct. 2010, pp. 265–278

[9] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. USENIX NSDI*, 2013, pp. 185–198.

[10] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency," in *Proc. 51st Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2013, pp. 731–738.

[11] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency," *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, Feb. 2016.

[12] D. Wang, G. Joshi, and G. W. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proc. ACM SIGMETRICS*, 2014, pp. 599–600.

[13] R. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999.

[14] M. Attia and R. Tandon, "Combating computational heterogeneity in large-scale distributed computing via work exchange," 2017, *arXiv:1711.08452*. [Online]. Available: https://arxiv.org/abs/1711.08452

[15] K. Shvachko *et al.*, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. MSST*, May 2010, pp. 1–10.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proc. USENIX HotCloud*, Jun. 2010, p. 10.

[17] D. Datla *et al.*, "Wireless distributed computing: A survey of research challenges," *IEEE Commun. Mag.*, vol. 50, no. 1, pp. 144–152, Jan. 2012.

[18] U. Drolia *et al.*, "The case for mobile edge-clouds," in *Proc. IEEE 10th Int. Conf. Ubiquitous Intell. Comput. Autonomic Trusted Comput.*, Dec. 2013, pp. 209–215.
[19] L. Song, S. R. Srinivasavaradhan, and C. Fragouli, "The benefit of being flexible in distributed computation," 2017, *arXiv:1705.08464*. [Online]. Available: https://arxiv.org/abs/1705.08464
[20] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1515–1529, Mar. 2017.
[21] R. Tandon, Q. Lei, A. Dimakis, and N. Karampatziakis, "Gradient coding," 2016, *arXiv:1612.03301*. [Online]. Available: https://arxiv.org/abs/1612.03301
[22] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2418–2422.
[23] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," 2017, *arXiv:1705.03875*. [Online]. Available: https://arxiv.org/abs/1705.03875
[24] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using Reed–Solomon codes," 2017, *arXiv:1706.05436*. [Online]. Available: https://arxiv.org/abs/1706.05436
[25] Q. Yu, M. Maddah-Ali, and A. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," 2017, *arXiv:1705.10464*. [Online]. Available: https://arxiv.org/abs/1705.10464
[26] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," 2017, *arXiv:1707.00040*. [Online]. Available: https://arxiv.org/abs/1707.00040
[27] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 964–971.
[28] S. Li, M. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," 2016, *arXiv:1604.07086*. [Online]. Available: https://arxiv.org/abs/1604.07086
[29] Q. Yu, S. Li, M. Maddah-Ali, and A. Avestimehr, "How to optimally allocate resources for coded distributed computing," 2017, *arXiv:1702.07297*. [Online]. Available: https://arxiv.org/abs/1702.07297
[30] Y. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: An alternative trade-off curve," 2017, *arXiv:1705.08966*. [Online]. Available: https://arxiv.org/abs/1705.08966
[31] S. Li, M. Maddah-Ali, and A. S. Avestimehr, "Coded distributed computing: Straggling servers and multistage dataflows," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2016, pp. 164–171.
[32] S. Li, M. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom*, Dec. 2016, pp. 1–6.
[33] M. Attia and R. Tandon, "Information theoretic limits of data shuffling for distributed learning," in *Proc. Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[34] M. Attia and R. Tandon, "On the worst-case communication overhead for distributed data shuffling," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2016, pp. 961–968.
[35] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Edge-facilitated wireless distributed computing," in *Proc. Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.
[36] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2643–2654, Oct. 2017.
[37] M. A. Attia and R. Tandon, "Approximately optimal distributed data shuffling," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 721–725.
[38] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, Mar. 2011.
[39] L. Song and C. Fragouli, "A pliable index coding approach to data shuffling," 2017, *arXiv:1701.05540*. [Online]. Available: https://arxiv.org/abs/1701.05540
[40] A. Elmahdy and S. Mohajer, "On the fundamental limits of coded data shuffling for distributed learning systems," 2018, *arXiv:1807.04255*. [Online]. Available: https://arxiv.org/abs/1807.04255
[41] K. Wan, D. Tuninetti, and P. Piantanida, "On the optimality of uncoded cache placement," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Sep. 2016, pp. 161–165.
[42] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 1613–1617.
[43] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, Feb. 2015.
[44] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 836–845, Apr. 2016.
[45] K. Shanmugam, A. M. Tulino, and A. G. Dimakis, "Coded caching with linear subpacketization is possible using Ruzsa-Szeméredi graphs," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*. Jun. 2017, pp. 1237–1241.
[46] L. Tang and A. Ramamoorthy, "Low subpacketization schemes for coded caching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2790–2794.
[47] M. Cheng, Q. Yan, X. Tang, and J. Jiang, "Coded caching schemes with low rate and subpacketizations," 2017, *arXivarXiv:1703.01548*. [Online]. Available: https://arxiv.org/abs/1703.01548
[48] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," 2018, *arXiv:1802.03389*. [Online]. Available: https://arxiv.org/abs/1802.03389

Mohamed Adel Attia (S'15) received his B.Sc. degree in electrical engineering from Alexandria University, Alexandria, Egypt, in 2012, and his M.Sc. degree in electronics and communications engineering from the American University in Cairo, Egypt in 2015.

He joined the University of Arizona as a Graduate Research Assistant in 2015. He is currently a Ph.D. candidate at the Electrical and Computer Engineering Department, the University of Arizona, Tucson, AZ, USA. His current research interests include Information Theory, Machine Learning, Wireless Communications, private information retrieval, and Distributed Data Analysis.

Ravi Tandon (SM'17) is an Assistant Professor in the Department of ECE at the University of Arizona. Prior to joining the University of Arizona in Fall 2015, he was a Research Assistant Professor at Virginia Tech with positions in the Bradley Department of ECE, Hume Center for National Security and Technology and at the Discovery Analytics Center in the Department of Computer Science. He received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Kanpur (IIT Kanpur) in 2004 and the Ph.D. degree in Electrical and Computer Engineering from the University of Maryland, College Park (UMCP) in 2010. From 2010 to 2012, he was a post-doctoral research associate at Princeton University. He is a recipient of the 2018 Keysight Early Career Professor Award, NSF CAREER Award in 2017, and a Best Paper Award at IEEE GLOBECOM 2011. He is a Senior Member of IEEE and currently serves as an Editor for IEEE Transactions on Wireless Communications. His current research interests include information theory and its applications to wireless networks, communications, security and privacy, machine learning and data mining.